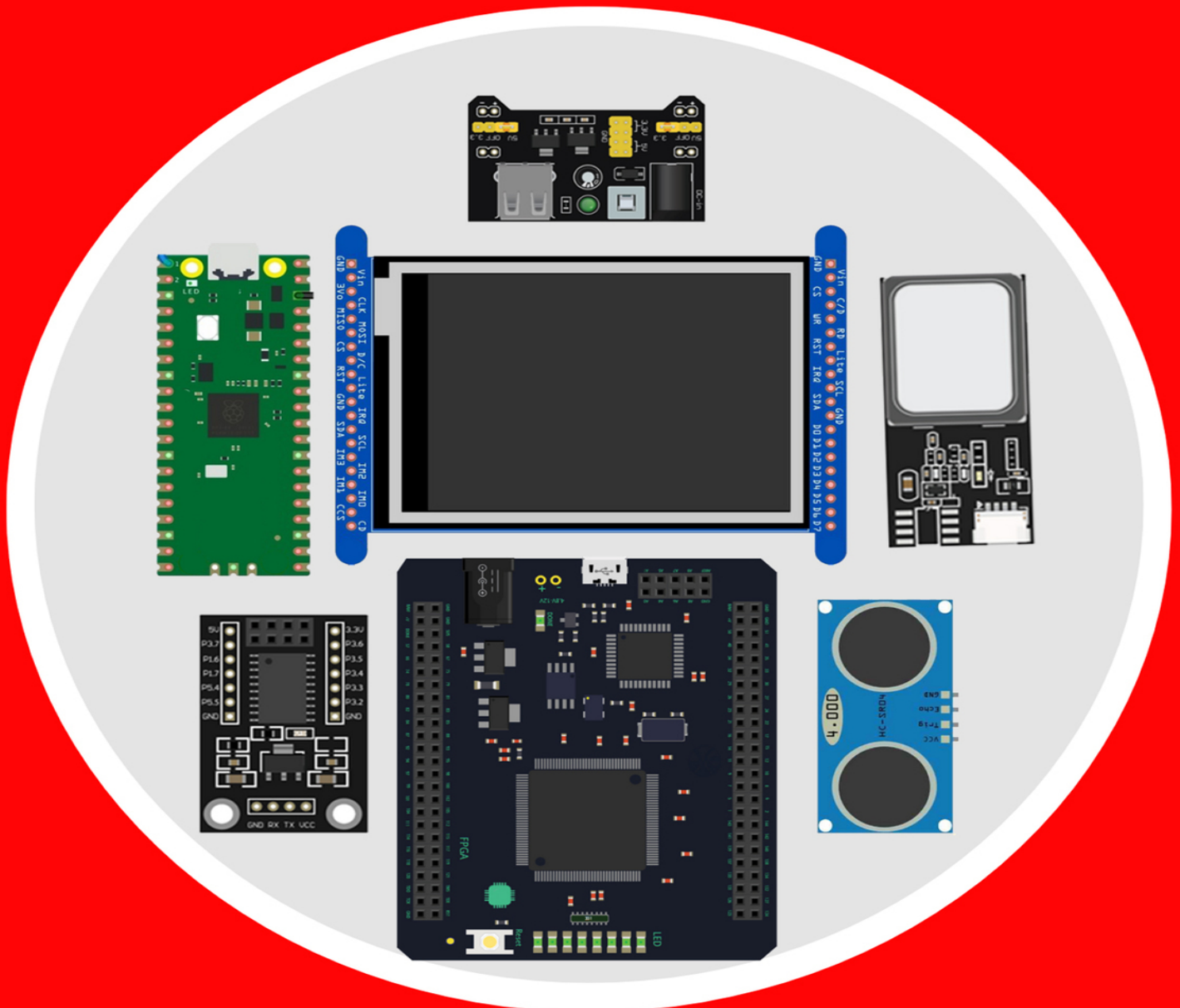


MASTERING FPGA EMBEDDED SYSTEMS

A Case Study Approach to Designing and Implementing
FPGA-Based Embedded Systems with TFT LCDs



MASTERING FPGA EMBEDDED SYSTEMS

**A Case Study Approach to Designing and
Implementing FPGA-Based Embedded Systems
with TFT LCDs**

By

Aharen-san

TABLE OF CONTENTS

[EDA TOOLS](#)

[WHAT'S EDA](#)

[EXAMPLES OF EDA TOOLS](#)

[TYPICAL FEATURES OF EDA TOOLS](#)

[SIMULATION](#)

[PCB DESIGN](#)

[SOFTWARE DEVELOPMENT](#)

[EDA TOOLS FOR FPGA DEVELOPMENT](#)

[THE USUAL USER STEPS](#)

[CREATE A PROJECT](#)

[WRITE YOUR CODE](#)

[PIN ASSIGNMENT](#)

[THE USUAL COMPILER STEPS](#)

[TIMING ANALYSIS](#)

[GENERATING A PROGRAMMING FILE](#)

[DOWNLOADING YOUR DESIGN INTO AN FPGA](#)

[QUARTUS PRIME, BY INTEL](#)

[ABOUT QUARTUS AND INTEL](#)

[SETUP_ DOWNLOADING QUARTUS](#)

[SETUP_ INSTALLING QUARTUS](#)

[WHERE EVERYTHING IS - PROJECT NAVIGATOR](#)

[WHERE EVERYTHING IS – TASKS](#)
[WHERE EVERYTHING IS - IP CATALOG](#)
[SETUP FOR IMPLEMENTATION](#)
[HELLO WORLD - CREATE A PROJECT](#)
[HELLO WORLD - TOP MODULE SOURCE FILE](#)
[HELLO WORLD – IMPLEMENTATION](#)
[HELLO WORLD - PROGRAMMING THE FPGA](#)
[EDA PLAYGROUND, BY DOULOS](#)
[ABOUT EDA PLAYGROUND AND DOULOS](#)
[EDA PLAYGROUND SETUP](#)
[WALKTHROUGH_CODE ENTRY](#)
[WALKTHROUGH_LEFT TOOLBAR](#)
[Walkthrough_Other Tools](#)
[DEMO_CODE EXAMPLE](#)
[DEMO_SIMULATION DUMP SYSTEM TASKS](#)
[DEMO_SIMULATION CONTROL SYSTEM TASKS](#)
[VIVADO DESIGN SUITE, BY XILINX](#)
[ABOUT VIVADO DESIGN SUITE AND XILINX](#)
[SETUP_DOWNLOADING VIVADO](#)
[SETUP_THE VIVADO INSTALLATION WIZARD](#)
[CREATING A PROJECT](#)
[WALKTHROUGH_PROJECT MANAGER](#)
[WALKTHROUGH_IP CATALOG](#)

[WALKTHROUGH_SOURCE FILES](#)
[WALKTHROUGH_CONSOLE OUTPUT AND MESSAGES](#)
[WALKTHROUGH_SIMULATION](#)
[WALKTHROUGH_PIN ASSIGNMENT](#)
[WALKTHROUGH_TOP MODULE CODE](#)
[WALKTHROUGH_BITSTREAM FILE GENERATION](#)
[WALKTHROUGH_THE HARDWARE MANAGER](#)
[LABSLAND](#)
[ABOUT LABSLAND](#)
[WALKTHROUGH](#)
[FPGA LAB](#)
[VERILOG IDE DEMO_PIN ASSIGNMENT](#)
[VERILOG IDE DEMO_ADDER CODE](#)
[VERILOG IDE DEMO_ADDER LIVE DEMO](#)
[VERILOG IDE DEMO_MULTIPLIER LIVE DEMO](#)
[RECOMPILING](#)
[UPLOADING DEMOS](#)
[MOTIVATION_HARDWARE DESIGN](#)
[MOTIVATION_SOFT PROCESSORS](#)
[INTRO TO FPGAS](#)
[FPGA OVERVIEW](#)
[FPGAS VS ASIC](#)

[WHAT'S INSIDE AN FPGA](#)
[WHAT'S INSIDE LOGIC BLOCKS _ ADDERS AND FLIP FLOPS](#)
[WHAT'S INSIDE INTERCONNECTS](#)
[WHAT'S INSIDE I _ O BLOCKS](#)
[WHICH IS THE PROGRAMMABLE](#)
[FPGAS VS CPLDS](#)
[HOW IS AN FPGA PROGRAMMED](#)
[WHO MAKES FPGAS](#)
[WHAT HARDWARE CAN BE IMPLEMENTED WITH AN FPGA](#)
[WHERE TO GET A BOARD](#)
[BOARD UNBOXING](#)
[THE DE0-CV BOARD WEBSITE](#)
[DE0-CV BOARD CD CONTENT](#)
[SKIMMING THROUGH THE MANUAL](#)
[THE FPGA DEVELOPMENT PROCESS](#)
[THE STEPS YOU NEED TO TAKE](#)
[CREATE A PROJECT](#)
[WRITE YOUR CODE](#)
[ASSIGN PINS](#)
[PIN ASSIGNMENT DEMO](#)
[SPECIFY TIMING CONSTRAINTS](#)

PROPAGATION DELAYS EXAMPLE
TIMING IN SEQUENTIAL SYSTEMS
WHY THIS MATTERS IN FPGAS
WHERE THE COMPILER TAKES ON
TIMING ANALYSIS
PROGRAMMING FILES
INSTALLING QUARTUS PRIME
SHOWING YOU AROUND QUARTUS PRIME
LOOKING AT THE TOP-LEVEL TEMPLATE CODE
ENTERING SOME PROOF-OF-CONCEPT CODE
COMPILING YOUR DESIGN
PROGRAMMING YOUR DEVICE - JTAG MODE
PROGRAMMING YOUR DEVICE - ACTIVE SERIAL
MODE
GETTING IT BACK TO ITS FACTORY STATE
SYSTEM BUILDER_ THE EASIEST WAY TO
JUMPSTART YOU APPLICATIONS
SYSTEM DESCRIPTION
LOOKING AT THE ADDER CODE
LOOKING AT THE BLINKY CODE
LOOKING AT THE INSTANTIATED MODULES
CODE
PROGRAMMING THE ADDER INTO THE BOARD

SCHEMATIC RTL DEMO

EDA TOOLS

First let me tell you about EDTA tools so in this project we'll get to know what the tools are.

In this lesson: EDA Tools

- What's EDA?
- Examples of EDA Tools
- Features of EDA Tools

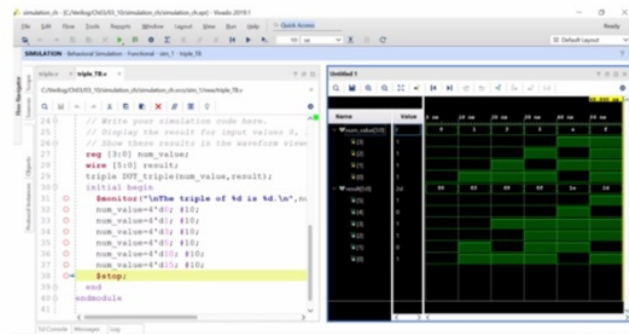
We'll see some examples of EDTA tools that are available out there and we'll get to know the typical features of EDA tools. So let's get started.

WHAT'S EDA

First what exactly is EDTA. Well the A stands for electronic design automation and that's what it means. It means automating the design process of any electronic devices by electronic we mean hardware usually. But it also supports software so these are software suites for creating hardware. That's a nice definition of PDA. They are similar to IV's which are integrated development environment for creating software. So you may think of Cold Warrior or Eclipse or Visual Studio.

What exactly is EDA?

- Electronic Design Automation.
- Software Suites for creating Hardware.
- Similar to IDEs.



These are integrated development environments but either way tools are something else but they are similar to ideas in that they have several modules that you can work with and you can make different projects with different objectives. So you are not completely expected to know how to use the whole suite but rather just the parts

that apply to your design. Of course you learn as you get more and more experience so here we have an illustration of the value Design Suite which is one of the tools you'll see in this project. And this is an FPGA designed too. There's a distinction between traditional IDEO tools and just FPGA tools.

EXAMPLES OF EDA TOOLS

So here we have some examples of EDA tools we have out young designer which specializes in hardware simulation and printed circuit board design. We have Eagle which also has a simulator but these two are mainly for designing PCBs.

Examples of EDA Tools

- Altium Designer
- EAGLE
- Vivado
- Quartus
- Multisim/Ultiboard
- Fritzing
- KiCad

Altium
Designer.



XILINX
ALL PROGRAMMABLE
VIVADOTM

We have the father Design Suite which is for FPGA. We have courts we have multi Sim an ulti board which are National Instruments tools for simulation and PCV design respectively there's a nice tool called fitting which is a simulator for creating both picked a graphical diagrams in a bread board and simulation Jacob is a very popular open source tool for PCV design.

TYPICAL FEATURES OF EDA TOOLS

So let's talk about the typical features of PDA tools. First we have schematic capture. This is where you enter a schematic diagram which is a graphical interface. We have HDL code entry with a text editor to enter very low or V HDL code.

Features of EDA Tools

- Schematic Capture.
- HDL Code Entry.
- Simulation.
- FPGA Design.
- Software Development.

Most of these tools have a simulator either analog or digital or mixed mode of course we also have FPGA design. This is the part we are interested in. We have software development and PCV design.

SIMULATION

So let me quickly summarize what simulation is in an easy way to. It means gathering data from the circuit models you are using in your design so that you can predict how the physical circuit would behave. Now there are several levels of simulation you can run so a simulation has the following elements.

Simulation

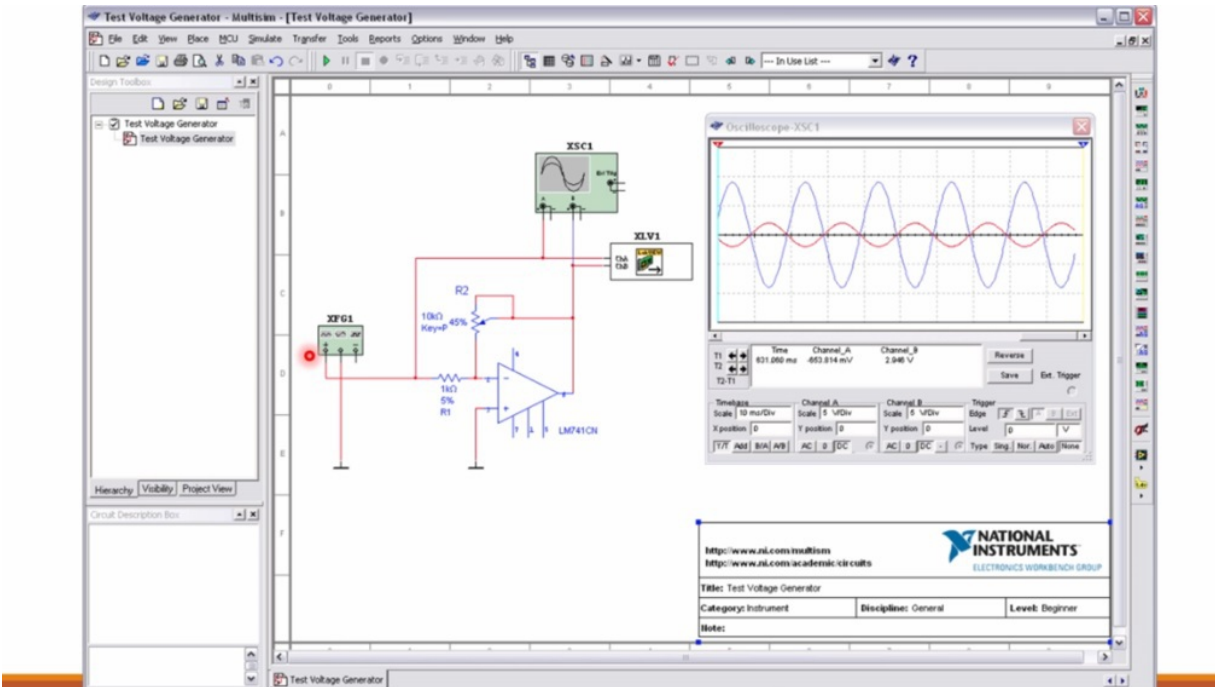
Gathering data from the circuit models in the design.

Typical elements:

- Digital Simulation
- Analog Simulation
- Mixed Mode Simulation
- Microcontroller & FPGA Simulation

We have digital simulation which deals with zeros ones high impedance and unknown states but that's it. It doesn't deal with voltages and currents. If you need those we have analog simulation which can simulate an amplifier for example or a low pass filter and finally we have mixed mode simulation. Let's say you have some logic gates in your design and you have some up amps and you want to see how it would behave in the real world. You can run a

mixed mode simulation which is actually the same analog simulation with analog models for the digital elements.



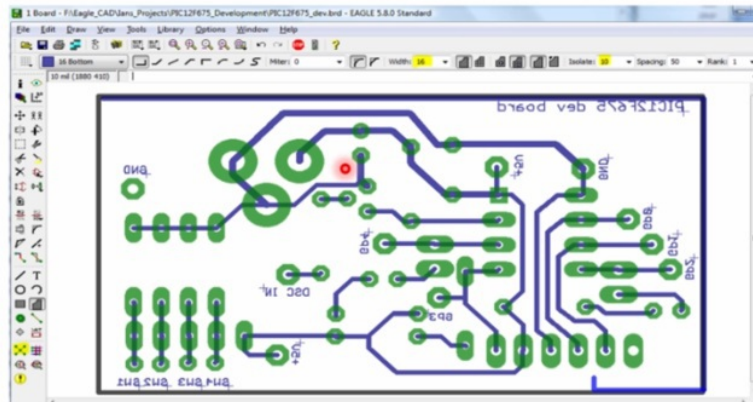
There is also microcontroller and FPGA simulation in case your idea too supports microcontrollers. It could have a simulator that works just as the microcontroller would work with your code so it requires a compiler tool usually. And as for FPGA simulation what we get is a simulator for the digital part. So I guess that would be the same digital simulation we just saw so here we have a nice simulation in multi Sim. This is National Instruments simulator and as you can see here we have an inverting amplifier with a part on the feedback resistor so we can change the gain. We have an oscilloscope at the output right here so we can see the signals and it's connected to both the input and the output in its two channels. So here's the result. And while the input comes from a function generator. So this is a nice example of a very popular analog simulator.

PCB DESIGN

PCV design is another feature of EDA tools which is very popular. Actually there are some tools that are only used for PCV design by their users and sadly most of these tools are underutilized because again people use them only to design pieces when they have a lot of other features. But anyway for PCV design what we need is routing of copper layers so remember a PCV has a lot of conductors which are printed on the board and these are layers of copper and the routing of copper layers is a very hard to solve computational problem. If you know about computational complexity you'll know about and beat problems. So this is an empty heart problem which is as hard as the hardest computational problem out there. And I'm mentioning this because this consumes a lot from your computer.

PCB Design

- Routing of Copper Layers
- SilkScreen Layers
- SolderMask
- Drilling
- Design Rules Check



If your PCB tool contains an outdoor router and I'm talking about a good auto router we also designed the silk screen layers which are the layers that have text or logos on them. These are usually white and this one for example says the number of the board right here. It says pick some number Development Board and notice that the text is flipped. That means that this text has to be printed on the other side of this board. We also designed the Solder mask which are the parts that are shown here in green. So these are the pads where Solder would go and PCBs usually have a coating of some material very similar to nail polish which prevents Solder from sticking into the tracks and only allows you to Solder where you are supposed to Solder. So these green parts are where the Solder mask would not go. So the Solder mask is the coating. And again this layer should be a negative layer which makes rings around these green pads where the coating is not supposed to go. We have drilling files that tells a drilling machine where to punch holes. So all of these pads are supposed to have drilling on them because this is a through hole PCB not a surface mount PCB and finally PCB designed tools also offer design rules check. So there are some guidelines provided by the manufacturers of PCBs because you are going to send your design to some factory to be built. Usually in high quantities and so these machines have some guidelines some requirements for example the clearance you need between elements and the clearance you need between the tracks and the borders. All of these rules are checked by the software in order to comply with the requirements of the manufacturing machines. You are going to use. You may build your own boards either manually or with some machine and even then these rules are useful.

SOFTWARE DEVELOPMENT

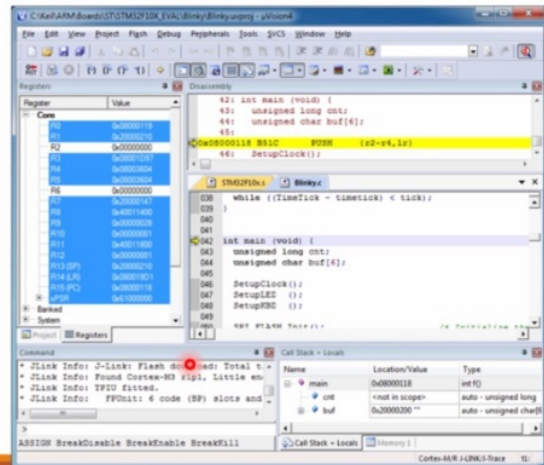
As for software development this is included in 88 tools mainly because of the microcontrollers or the soft processors. Your design may contain some elements of software development tools are actually the same ones found in 80s. So we have a text editor with usually syntax highlight to create your code and show it to you later. We have a compiler assembler a linker the works a compiler tool chain we usually have a D assembler so that you can see that this assembly as seen in this picture right here where you can see how your source code resulted in assembly code.

Software Development

Many EDA tools have software dev tools for the microcontrollers or soft processors in the design.

Typical Elements:

- Text Editor with syntax highlight
- Compiler/Assembler/Linker
- Disassembler
- Debugger



We also have a debugger and in this picture we have Kyle tools showing exactly that the debugger. So here are the registers of an

arm cortex CPA. Here's the source code and here's the disassembled code. Here we have some variables that can be shown to you in runtime. And this debugger usually contains a simulator so you can simulate whatever you're designing or you may download your design to an actual microcontroller. And from the microcontroller report back to the I.D. to show you the same debugger but with real data in these registers or memory location watches.

EDA TOOLS FOR FPGA DEVELOPMENT

Now what we are most interested in are LDA tools for FPGA development so in this project we'll talk about just that idea. Tools for FPGA ace. We'll talk about the usual user steps the steps you need to take in order to create your FPGA applications.

In this lesson: EDA for FPGAs

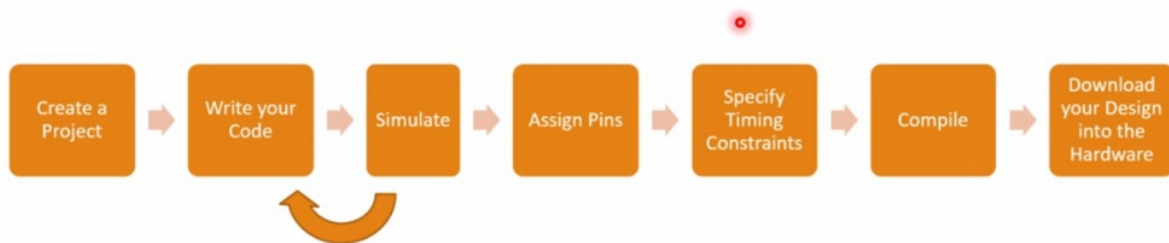
- The Usual User Steps
- The Usual Compiler Steps

We'll talk about the usual compiler steps which are the steps taken by the software and that's it. Now I want to make a little disclaimer please bear with me in this project because it contains a lot of the elements we already saw in the previous projects and I think that it's not only important that you keep all of this in mind but I will also mention some new important details so let's get started.

THE USUAL USER STEPS

Here are the usual user steps the steps you need to take to create your FPGA designs. So first you create a project. Then you write your code and then you usually simulate and there's a loop here. An explicit loop I have created here so that you can simulate and when you find out that your circuit is not behaving as you want it then you rewrite your code or modify your code rather simulate again modify your code.

The Usual User Steps



And when you're happy with your design you can go ahead to the next step. So the next step would be the implementation if you want to get your design into NAFTA. Then you have to assign the pins in the FPGA package then specified timing constraints which are your

timing requirements. The most basic of these is the clock frequency. You want to support. Then you compile which is what the computer does on its own. And finally you can download your design into your hardware that is your FPGA or most likely your development board. Now let me quickly tell you some details about all of these steps.

CREATE A PROJECT

The first step is to create a project so ideal tools usually work with projects. And so you have to provide some information for the tools to know how to create your application. So first we have the purpose. You may simply want to simulate your design or you may be interested in getting it on an FPGA. So as we just saw simulation requires about three steps. But FPGA design requires many more if you want to design for a target FPGA.

Create a Project

- EDA tools usually work with projects.
- Information you provide:
 - Purpose (Simulation or FPGA)
 - Target FPGA
 - Target Board (If supported)
 - Primary Source Language/Method
 - Source and Libraries (IP Cores)

You have to specify which chip you are using the target board if your tool supports several implementations of the same FPGA on different boards. Well you get to specify it. Now this is usually kind of optional but you get sometimes to specify which is the language you intend to use for code entry. Very Verilog v HDL but most tools are

bilingual and you usually get to select which IP project that's third party source or libraries that you get to include in your project.

WRITE YOUR CODE

The next step is writing your code and once more. Most idea tools support very low end VH deal some tools support schematics but these schematics aren't really for writing your code. These are usually for showing you the net list created by the Synthesis tool. By that I mean what your tool interpreted what it understood from your source code in very low or VH deal. So these schematics are for you to validate that the tool got it. The tool understood what you intended to create. There are two types of schematics shown for example in the other.

Write your Code

- Most EDA tools support:
 - Verilog
 - VHDL
 - Schematics (RTL and Technology)
- You have to include all modules in your project.
- Lots of IP-Cores available.

We have RTL schematics and technology schematics. The RTL schematic is a logical schematic which may show you gates and multiplexes and flip flops or even decoders whereas the technology schematic is limited to the technology available inside the Target

FPGA chip. So let's say your FPGA contains Gates. Then the RTL schematic will show you those gates and the technology schematic will also show you those gates. But on the other hand if you're FPGA as most FPGA does not contain free gates for you to use but rather lookup tables then the technology's schematic will show you. Lookup tables and going back to writing your code you have to include all modules you are going to use in your project. This is just the same as with software development but you don't have to write the source for all of the modules you're going to include. You can use IP project that are available from the vendors and many of those IP project are available for free.

PIN ASSIGNMENT

When it comes to assigning pins the EPA tools that support implementation usually allow you to enter some important information about these pins so you get to specify the location of the pins that is which physical pin in the FPGA chip goes to which input output brought off your top level module.

Assign Pins

- Some important information about pins is necessary:
 - Pin Location
 - Pin Direction
 - Schmitt Trigger
 - Pull Resistors
 - Slew Rate
 - Many, many more!

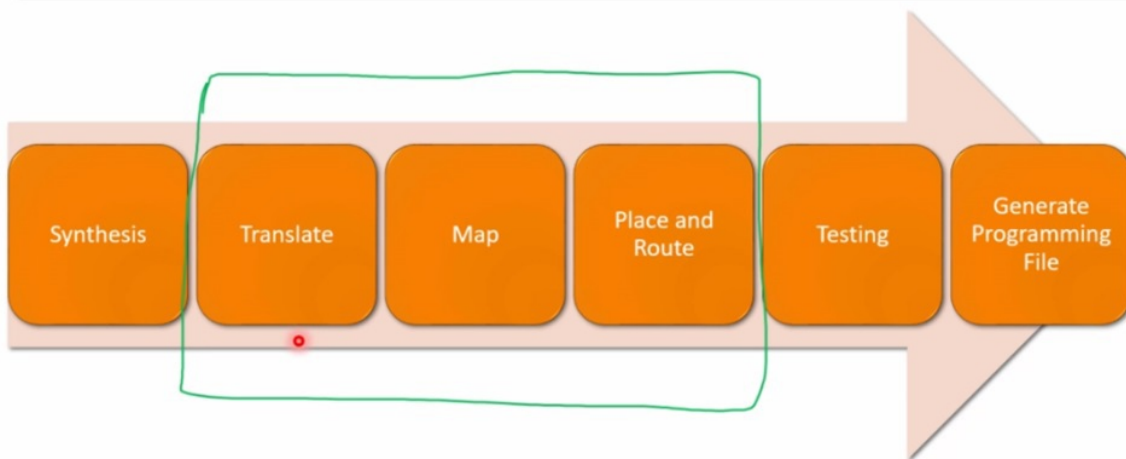
- This info resides ultimately in some text files.

You can indicate the direction of the pin either it's an input pin an output then or a bi directional pin your inputs may be Schmidt trigger they may have pulled resistors either pull up or pull down resistors you may specify even analog details about them for example slow rate and there are many many more aspects you may specify of Europeans all of this may vary from idea a tool to EDI to but this information usually is stored in some text file that you may modify.

THE USUAL COMPILER STEPS

No the compiler steps are synthesis. Translate map place route testing and generating a programming file. These are the steps shown in the site links tool chain. That is the one used by the vato Design Suite and their names may vary from tool to tool. Actually these may have different names in quotes for example so synthesis means converting your RTL design into a net list representation.

The Usual Compiler Steps



Like I said this means what the compiler understood from your very logger VH DL source code implementation means three steps translate map place and root. So going back to the flow diagram we can see these steps which are shown right here. Translate map place and route. So once again these three steps are what's usually

known as implementation. This is what the compiler does to create your design for an FPGA. The difference is that synthesis is performed for both implementation and for simulation. So when you simulate the compiler has to understand what you mean with your code. So synthesis is always performed and then the implementation has these three levels once again these three steps are named differently between platforms. So roughly translate means getting the net list into some code that the FPGA can create.

Implementation

- Translate
- Map
- Place & Route

So the translation may produce an intermediate representation of your hardware then map consists on assigning the intermediate elements to the available elements in your FPGA and finally placing root means selecting which elements which were mapped are going to go where in your FPGA and routing between these elements to create the already mapped design. You don't have to do anything about these steps and they may output some errors or warnings which you may have to correct in your code but this is all the responsibility of the compiler. You simply have to sit back let it work and see if the output prompts you to do some changes.

TIMING ANALYSIS

There's a timing analysis that the tools often enabled you to perform and this is to test the implementation you just made.

Timing Analysis

- The implementation steps take timing constraints as rules
- The generated circuitry is analyzed for compliance with constraints



So when your design was implemented these implementation steps will have taken into account the timing constraints you must have entered and these timing constraints are considered rules for the implementation. So the generated circuitry is then analyzed by the tool to verify that it compliance with all of these constraints.

GENERATING A PROGRAMMING FILE

One of the final steps is generating a programming file. Intel calls this assembly code but it's not the typical assembly code in microprocessors sailing schools it bit stream which is I think a better name. This is only a binary file. This file may go to the FPGA which usually has a RAM memory to store this design.

Generating a Programming File

- Intel calls it "Assembly".
- Xilinx calls it "Bitstream".
- This is the binary output.
- Goes to the FPGA, or On-Board Flash.

You know what that means. That means that when you turn off the power then the design is lost. So it may also go to a onboard flash and FPGA are usually designed that way. The chip has a RAM memory to store the design in the FBI and it's usually required to have a supporting flash memory onboard to dump that design into

the FPGA on startup so let me tell you a bit more about this last point.

DOWNLOADING YOUR DESIGN INTO AN FPGA

Downloading your design into the FPGA means that it goes to a RAM memory because most FPGA A's have a RAM memory where they store your design. You know what that means don't you. That means that this is volatile. So when you turn off the power the design is lost. Does that make sense. Well it makes sense in that Ram memories very fast to use and to program so downloading to the FPGA directly is convenient for development because when you are developing your application it will load faster and you're usually not interested in keeping that design in your FPGA anyway.

Downloading your Design

INTO THE FPGA

- Usually in RAM.
- This is Volatile!
- Convenient for development.

INTO A NON-VOLATILE MEMORY

- Downloaded at power-on.
- This is part of the boot-up procedure.
- Convenient for deployment.



Now when you download your design into a non-volatile memory that's usually on the same board as the FBA. Well that gets

downloaded into the FPGA at power on notice that when you program this memory it will take longer than when you programmed the RAM memory so that's not so convenient for development and this download process that goes on from the RAM memory into the RAM memory that's into the FPGA. This is all part of the boot up sequence of the FPGA. So any applications you know like your TV or your smartphone or whatever electronics you have that contains an FPGA if that FPGA implements his memory in RAM it must have some non-volatile memory onboard that as part of the boot up procedure dumps this design from non-volatile memory to Ram and finally. Well this is convenient for deployment when you want to ship your designs you programmed the flash memory so that every time the user turns on the device your design will be loaded into the FPGA at power on.

QUARTUS PRIME, BY INTEL

Now it's time to talk about Quartus Prime so let's see what we'll talk about in this project. First I'll tell you about Quartus Prime and Intel. I'll tell you about the setup that's required to install and get your environment up and running.

In this lesson: Quartus Prime

- About Quartus Prime and Intel
- Setup
- A Quick Walkthrough
- Code Example
- Simulation
- Implementation

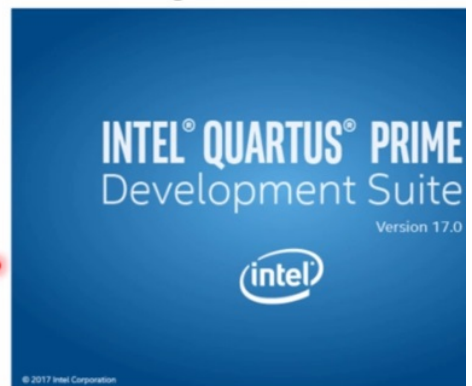
I'll give you a quick walkthrough of this PDA too. I'll show you a code example then a simulation. And finally an implementation. That's because I will show you all of the steps that are supported by this very powerful tool. So let's get started.

ABOUT QUARTUS AND INTEL

So let me tell you about Curtis and Intel. CURTIS Prime is a full-fledged FPGA development tool. By that I mean that it supports all of the steps we have just mentioned for FPGA development. So this includes simulation and implementation. You may remember that this LDA tool is featured in our second project. The basic FPGA training so you may have some practice on this tool. But anyway I hope you learn some new tricks as you see the examples I'm about to show you now.

About Quartus Prime and Intel

- Quartus Prime is a full-fledged FPGA development tool.
- Featured in our second course: **Basic FPGA Training**.
- It offers free licenses.
- Setup starts at download.
- Made by Intel, who acquired Altera.



Although Quark This is a commercial application Intel offers some free versions and licenses that are free of cost and these usually have some classes that require you not to charge for your designs

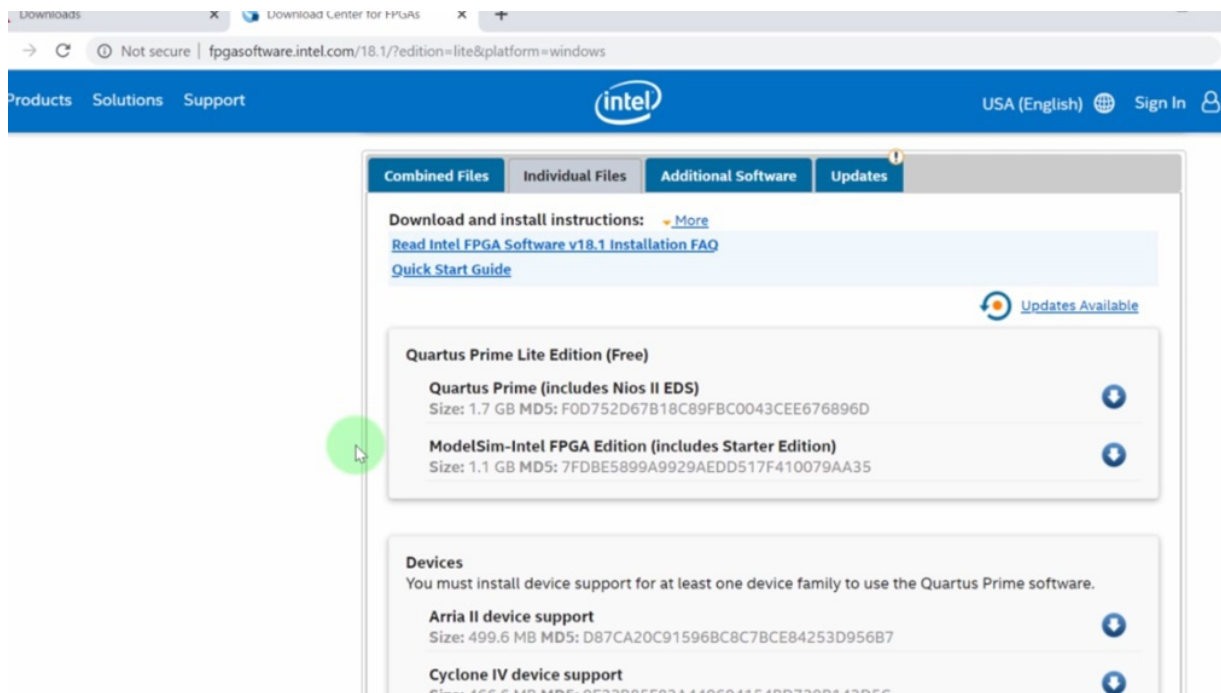
and these are also limited in this size of the target application. But these licenses work pretty well for development boards for hobbyists and definitely for discourse. Now the setup starts at download. You get to download several modules into your installation of quarters and Intel makes it easy for you to choose which elements you want to include in order to make both your download later and your installation process faster needless to say. This is an Intel product but you may also know that until recently acquired Altera which was the original company that developed both Quark Does Prime and the manufacturer of all Altera FPGA. So if you were familiar with ultra well now it's Intel. It's been some years now since the acquisition but you may still see the Altair a name out there in some FPGA is.

SETUP_ DOWNLOADING QUARTUS

So the first step in using cordless prime of course is downloading it and so you can make a web search. Here I have a Google search for cactus. It's not a very common name. So the first result is FPGA design software. Intel called the spring. Just make sure you go to Intel's Web site. So this is the core. This Web site if you scroll down I'm sure you can find the download space which is here features and download and we have several versions. So we have the prime pro edition the prime standard edition and the prime Late Edition. This is the one you want if you want it for free. So this is the download page and I think it requires some explanation.

The screenshot shows the Intel Quartus Prime Lite Edition download page. The browser address bar shows the URL `fpgasoftware.intel.com/?edition=lite`. The page header includes the Intel logo and navigation links for Products, Solutions, Support, and Sign In. The main content area is titled "Download Center for FPGAs" and features a sidebar with navigation options: Design Software, Embedded Software, Archives, Licensing, Programming Software, Drivers, Board System Design, Board Layout and Test, and Legacy Software. The main content area displays the "Quartus Prime Lite Edition" product page, which includes the release date (September, 2019) and the latest release (v19.1). The page offers a selection of editions (Lite and Pro) and operating systems (Standard, Windows, and Linux). A note indicates that the Windows version is not yet available. A yellow box at the bottom contains three green checkmarks: 1) The Quartus Prime Design Software Lite Edition v19.1 includes functional and security updates. 2) OS Support and IP support have changed in this release. 3) The Quartus Prime Lite software version 19.1 supports the following device families: Arria II, Cyclone 10 LP.

So first you get to decide which edition you want. We have the pro the standard and the light editions. Once again we want the light edition and you get to select the version the release. So as you can see we get from nineteen point one which is the latest all the way down to version 3. So as always I recommend the latest version. But there's a little problem with this one and that's if you are using windows you can't select windows for this one because as this note says it's not ready yet since I'll be working with Windows.



I'll work with the earlier version if you're working with Linux and you want to use release nineteen point one You are free to do so. But I will select eighteen point one as you can see now I can select windows and scrolling down you will see all of the elements you can download. So right here you get to decide if you want to download all of the files combined into one single download which is not usually recommended because of the size of the download. This is five point eight gigabytes. And the default is individual files in this tab right here. And you'll notice that there are several separate elements. It's not that complicated. You just have to download the executable files

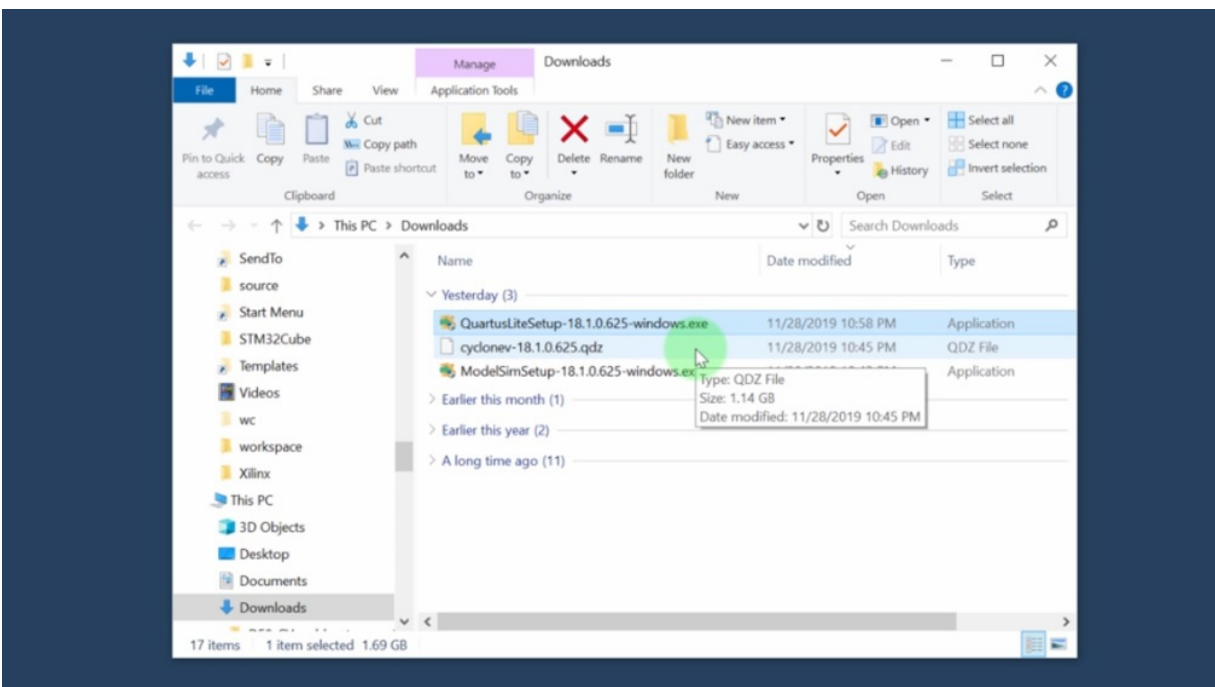
you want. In my case I will download quite as subprime which is the FDA too.



And I will download model Sim which includes the FPGA edition and the Starter Edition. One of these two is the one you are interested in and finally you get to decide which devices you want your installation of characters to support. So Altera or now Intel has several families of FPGA. One is called Aria. There's the cyclone family with several versions. The max family. So since I will be using the DS Development Board let me show you. This is the board I'll be using. This is the D 0 CV board where CV stands for cyclone 5. Then I want to support the cyclone 5 FPGA base and I'm sure you can distinguish a cyclone right here which is the logo for the cyclone five FPGA is so this board has a bunch of IO some switches some LCD some segment displays buttons and a connector with access to virtually every pin in the FPGA. So going back I will select cyclone five device support if you have any other development board and you know the FPGA it has on it then you should select to download support for that FPGA as well.

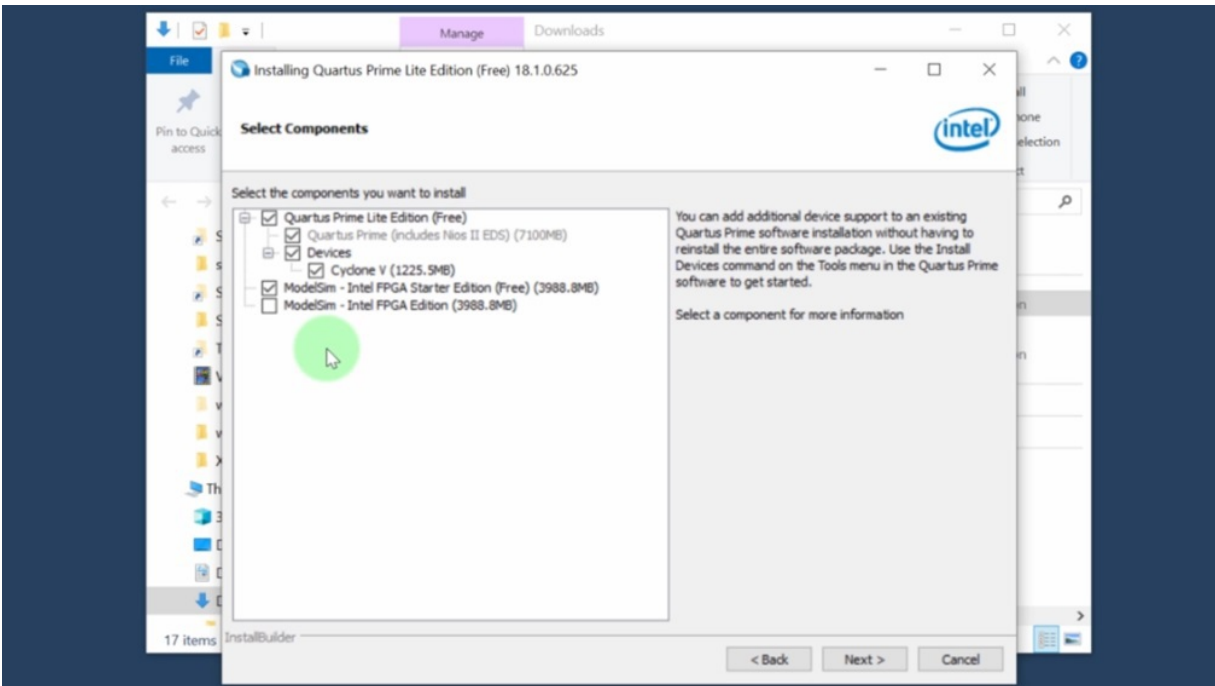
SETUP_ INSTALLING QUARTUS

So here we have what I've downloaded. Here's the installer for Curtis. Here's the support package for the cyclone five FPGA and this is the model Sim installer. Now the quietest does installer is smart in that it will check in the same directory for the other installers.



You may have downloaded and it will offer you to install them along with quarters. So let's don't click on the quietest light setup so first. Just make sure that you're installing the free edition. That's the LATE EDITION. Next we have to agree to the terms you get to choose the directory where you'll install and this is the part I told you about where you get to choose which elements to install. Now the

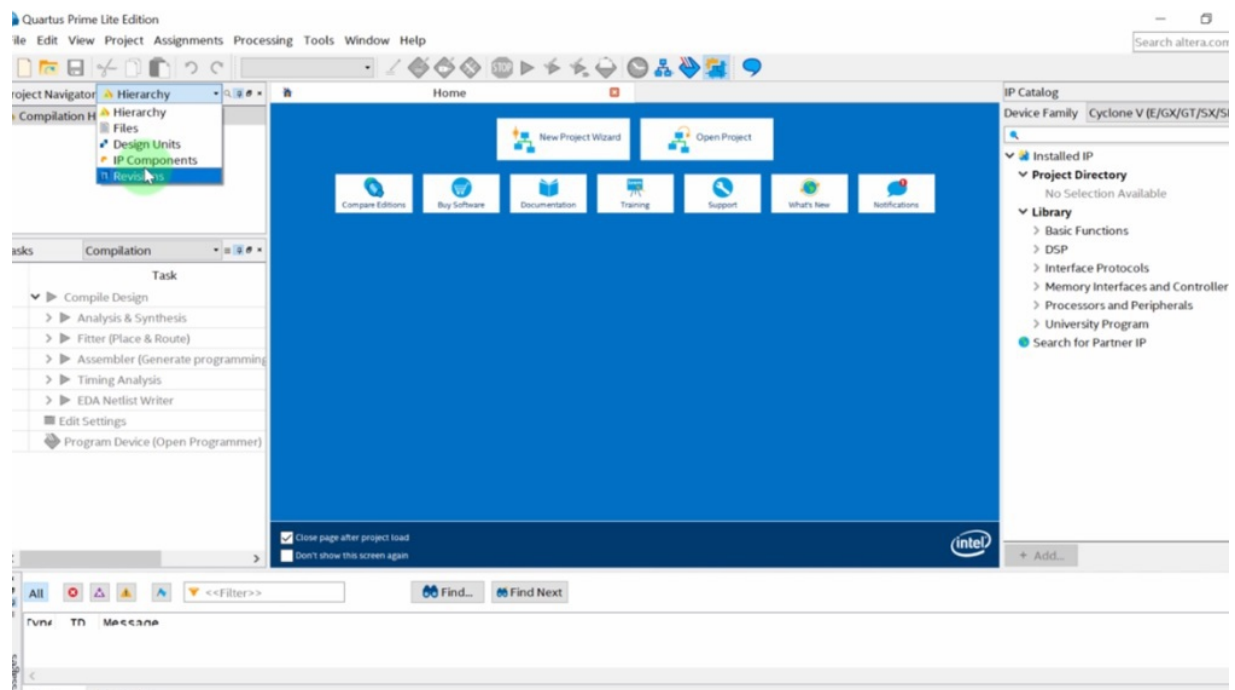
elements that appear here are based on the other installers you've downloaded for example here we have the quotes to subprime Late Edition installer but then we have devices here and only one is shown which is the cyclone 5 and that's the download. I have along with Curtis right here.



Notice that I have models him also in this directory so model same is shown here. Now the default is to install the Starter edition of model Sim which is set to be free and we also get to install another edition that doesn't see starter just until FPGA edition. So it seems this is all we need and we are okay with these default settings so we are okay to go to the next step which is a summary and we get to install now. And as you know this may take a while once it's done you get the usual stuff among which we have to launch the USP blaster to driver installation. So this is a device driver for the hardware downloader into many of Intel's boards. So you want to do this and here you just have to follow along finally you get to choose the license if you want to buy a license or just run Cordis. Remember that this is a late edition so you get to run it for free.

WHERE EVERYTHING IS - PROJECT NAVIGATOR

So once installed quarters is ready to run any simulation project you want. So this is the welcome screen where you get to choose which project to open or to create a project and all of the things we have in these buttons right here. So right now there's no project open but let me show you where everything is so he read the left we have the project navigator. Where do you get to see all of the data regarding your open project.

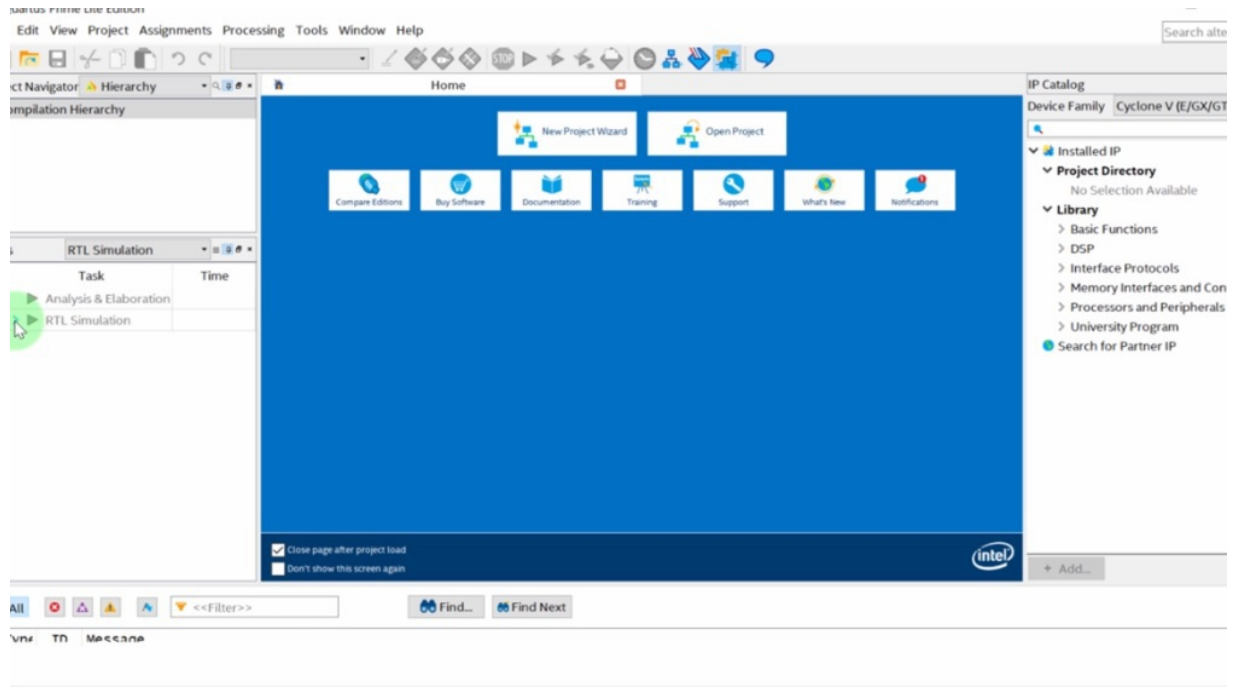


And here we have a dropdown where you get to choose the aspects you want to see in the project navigator regarding your project. For example here we have hierarchy. You can see the files the design units the AP components and the revisions of your project. So it

comes down to what you want to work with which level of abstraction you want to work if you want to simply see the files you can select files if you want to work with your units will you select hierarchy. That's why this is the default.

WHERE EVERYTHING IS – TASKS

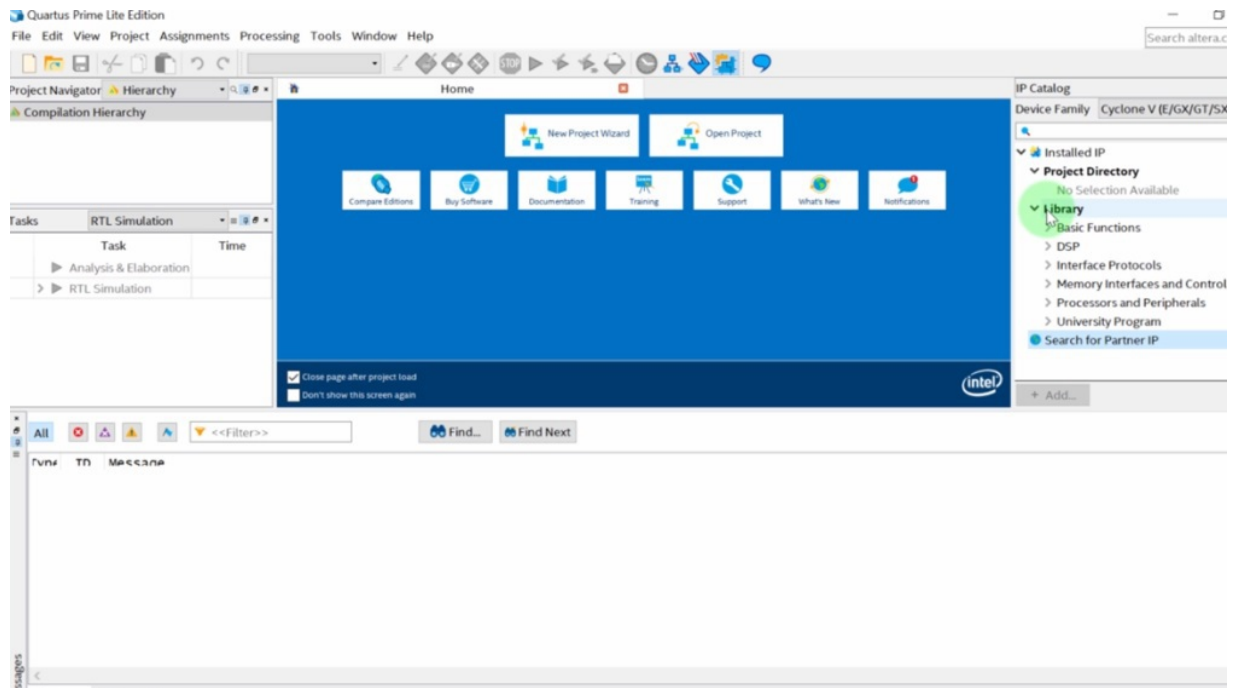
Next we have the task section here where you can find all of the steps of the synthesis or implementation process you want to perform. And like I mentioned before these steps depend on the objective of your project. So you get to choose that objective in this dropdown right here. So this is for compilation but we can also choose a full design a Gates level simulation or RTL simulation or a rapid recompile. And if you choose any of these elements you will see that the steps shown in this list will change. For example this is for a full design. And here we get compile design we get verified design export database archive project and as a final step in implementation we are usually interested in the programming file. So that's what this step is for. It says program device open programmer going back to the objectives of your project. We get a great level simulation where you get to finally run a simulation. We get to perform timing analysis and we get some steps all the way down to Gates level simulation.



However with the RTL simulation we get only one step after analysis and elaboration which is RTL simulation. Again the difference between RTL and gate level is that Gates level takes the lowest elements into account whereas RTL will base only the semantics of your very log or VH HDL design. Moving on at the bottom we have what we usually have in any idea which are the output sections. For example here we have the error messages warning messages some more warning messages. So these are critical warnings and these are all warnings and you also get to choose which messages you are seeing. So that's what these taps right here are for. Next we have at the middle. These buttons right here which are simply shortcuts to perform several operations since we don't have a project yet. This is like the welcome screen. It's called home. So we get to create a new project. We could open an existing project and we get a bunch of tools right here to assist you in your development process.

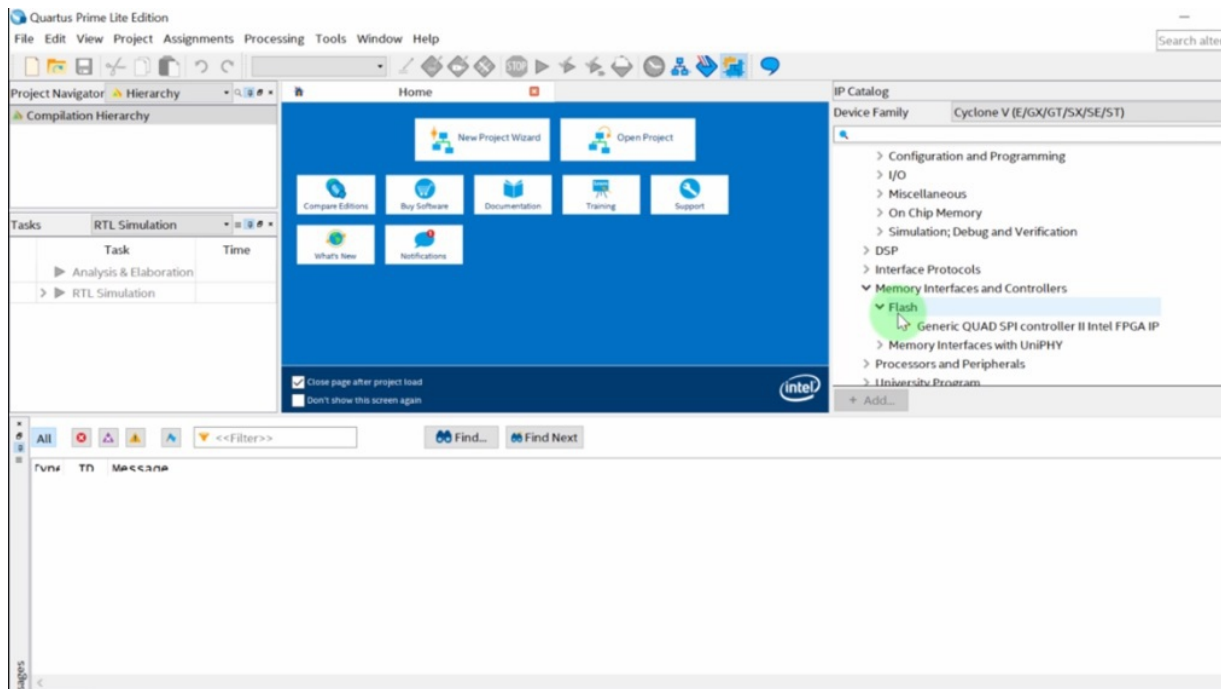
WHERE EVERYTHING IS - IP CATALOG

At the right we have the AP catalog and this is very very cool. So you get to choose all of the AP that's available to you either with a license that's commercial IP or open source IP or even educational IP. So the structure goes like this. You have a top category right here that's called installed IP. You could search for partner IP this takes you to the intel Web site and you get to perform a search for IP. So there's a wide variety of elements available let's go back and looking at the installed IP you'll see that there's already a lot of IP available in the installation.



So here we have a project directory since we don't have any project open. Well nothing is shown here and then we have a library. So

here we have basic functions arithmetic bridges and adapters clocks you name it. So let's see for example arithmetic we have let's say Altera floating point functions multiply other intel FPGA IP etc. We have memory interfaces and controllers for example here's something for a flash memory.

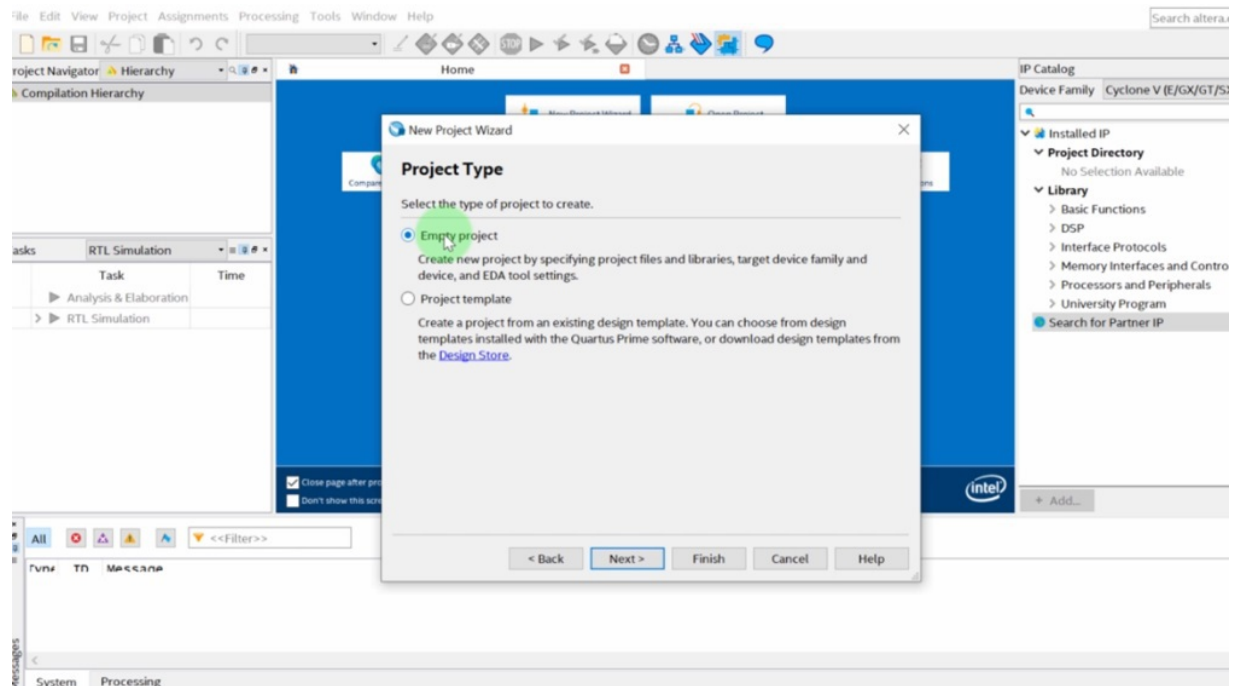


This is a generic quad spi controller but what I want to show you are these two elements processors and peripherals which are soft processors. So here we have code processors so here we have NEOs to custom instructions so here we have instructions you could make on your own for the NEOs too. Next we have heart processor components where you get to implement peripherals for heart process source that are included in your boards. And here we have peripherals. So we have a night to see slave an interrupt controller. And if all of this seems very limited to you. Well we have the university program section right here where we have a bunch of stuff for you to use freely with educational purposes. So here we have audio and video elements. We have clock elements communications generic IO and as you can see we have a U.S. B controller here for example. So you don't have to create your own and other memory

we have a nice room controller. So this is a very nice library to start working with I.T..

SETUP FOR IMPLEMENTATION

So if you want to create your implementation project to download your device into an FPGA you will need to do some setup. And for that you can ask to create a new project. Here we have some instructions and here you have to enter some things first your working directory. Second the name of your project. And finally the top level design entity of your project. That's the module that will be either downloaded to your FPGA or the top simulation module so let me create a directory called EDTA tools quarters. Let's name this project. My D 0 CV. Why. Because I'll be using a D 0 CV bought automatically the name of the project is copied to the top level module.

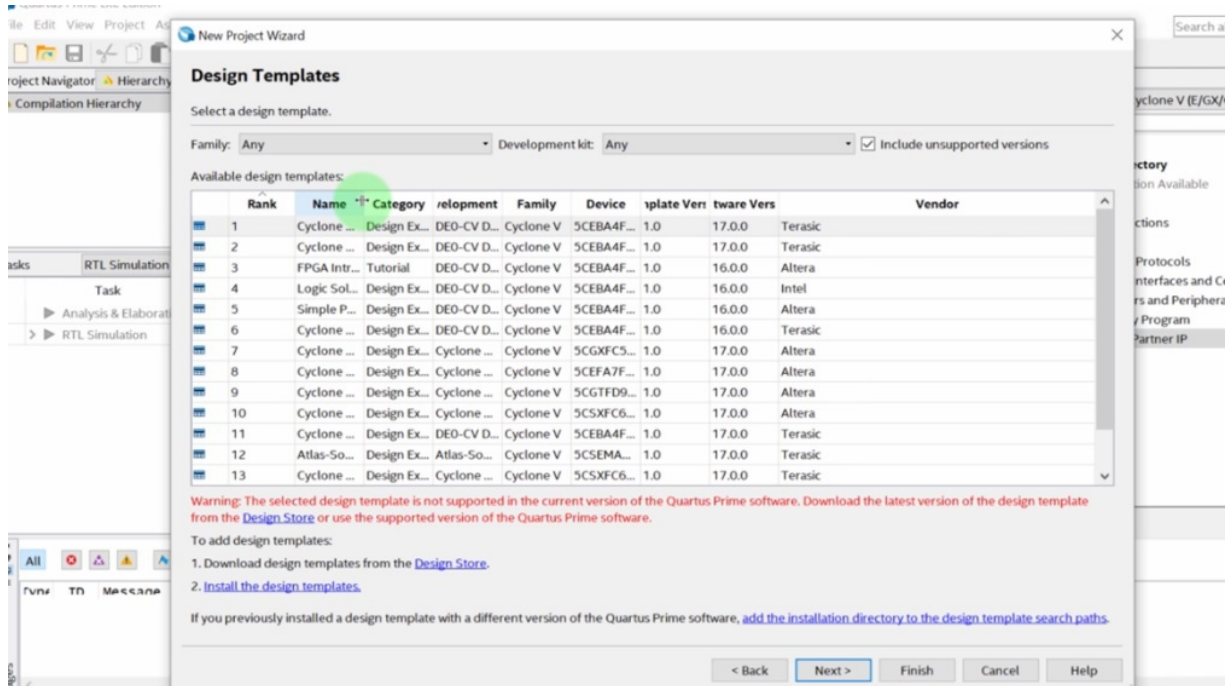


I'm okay with that. So this is where you get to download your templates for the board you are using if you want support for a development board so you can always create an empty project where you'll or all of the data by yourself. This will be kind of an advanced thing for you. So if you want to support your board for example that the e 0 CV by thoracic then I recommend you choose project template. Now let me read this part. It says create a project from an existing design template. You can choose from design templates installed with the quarter spraying software or download design templates from the design store. So if we click Next we'll see the design templates that are available in your installation. These are most likely up to date until the latest update you have performed on your software. So that's not always the best choice so I recommend that you go to the design store and download the latest support for your board. So let's see this link design store. This will take you to the intel Web site again. And here you can look for your board. So let's look for my these zero board which is a cyclone five. Let's see the categories available. We have designed examples and tutorials. Let's leave it at any.

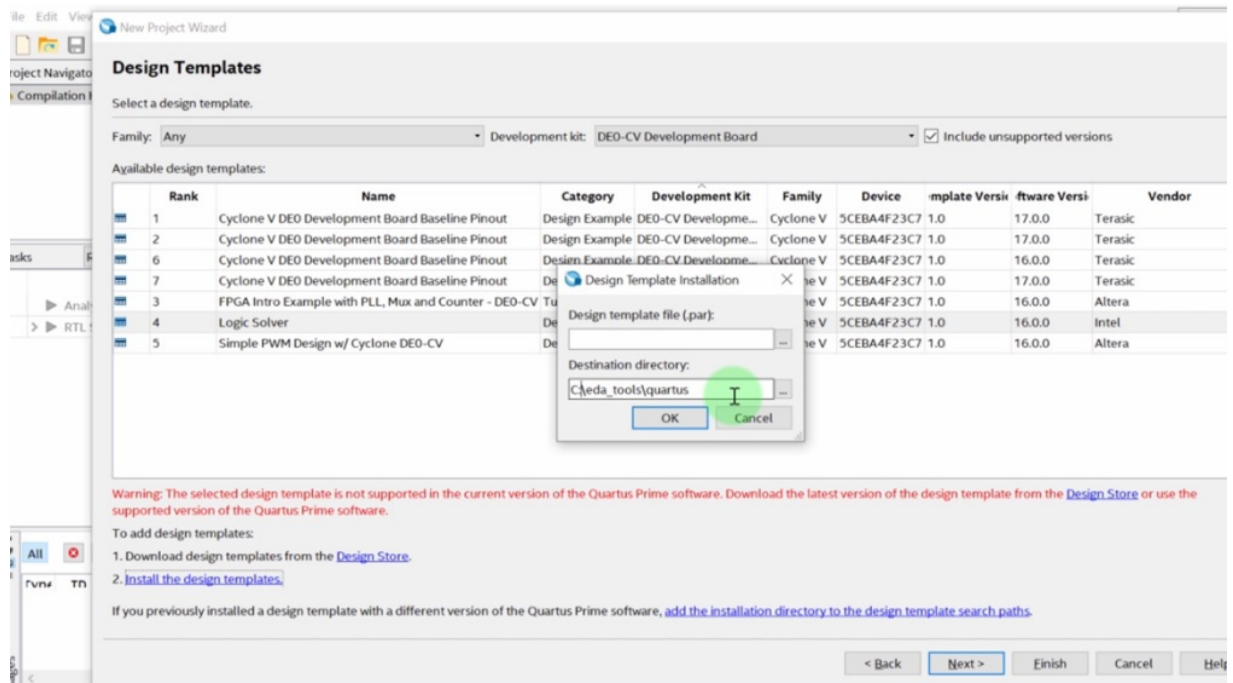
The screenshot shows the Intel Design Store interface. At the top, there are navigation links for Products, Solutions, and Support, along with the Intel logo, language selection (USA (English)), and a Sign In button. Below the navigation is a search bar and a search button. The main content area features a filter section with dropdown menus for Family (Cyclone V), Category (Any), Quartus Prime Version (Any), Development Kit (Any), and IP Core (Any). Below the filters is a table of design examples.

| Name | Category | Development Kit | Family | Quartus Prime Version | Vendor | Downloads |
|-------------------------------------------------------------------------------------|---------------------------------------|------------------------------------------------------------|-----------|-----------------------|--------|-----------|
| Gen2x4 AVMM DMA - Cyclone V | Design Example \ Outside Design Store | Cyclone V GT FPGA Development Kit | Cyclone V | 14.0.0 | Intel | 66 |
| Sigma-Delta Converter - Cyclone V | Design Example \ Outside Design Store | Non kit specific Cyclone V Design Examples | Cyclone V | 9.1 | Intel | 11 |
| AN 307: Altera Design Flow for Xilinx Users - Cyclone V | Design Example \ Outside Design Store | Non kit specific Cyclone V Design Examples | Cyclone V | 12.1 | Intel | 4 |
| AN 456: PCI Express High Performance Reference Design for Cyclone V | Design Example \ Outside Design Store | Non kit specific Cyclone V Design Examples | Cyclone V | 14.0.0 | Intel | 22 |

Let's see the quarter subprime version. We are using eighteen point one. And here we get to choose our board. So let's look for the easier aero CV development board right here and we also get to choose a knight core. We are not looking for anything in particular. So that's all we need. And here it is. It's this logic solver. This is an example. And it's for the DRC v Development Board. So all you have to do is click here on the name of the element and you'll get a download link. Right here. Notice that if we go back to quarters without downloading this we'll always get this part of the Wizard where you'll see the installed elements. Let's look at it at this point let's click next. So here we can see the supported elements. Let's see if we have the D 0 CV. Yes. Here it is.

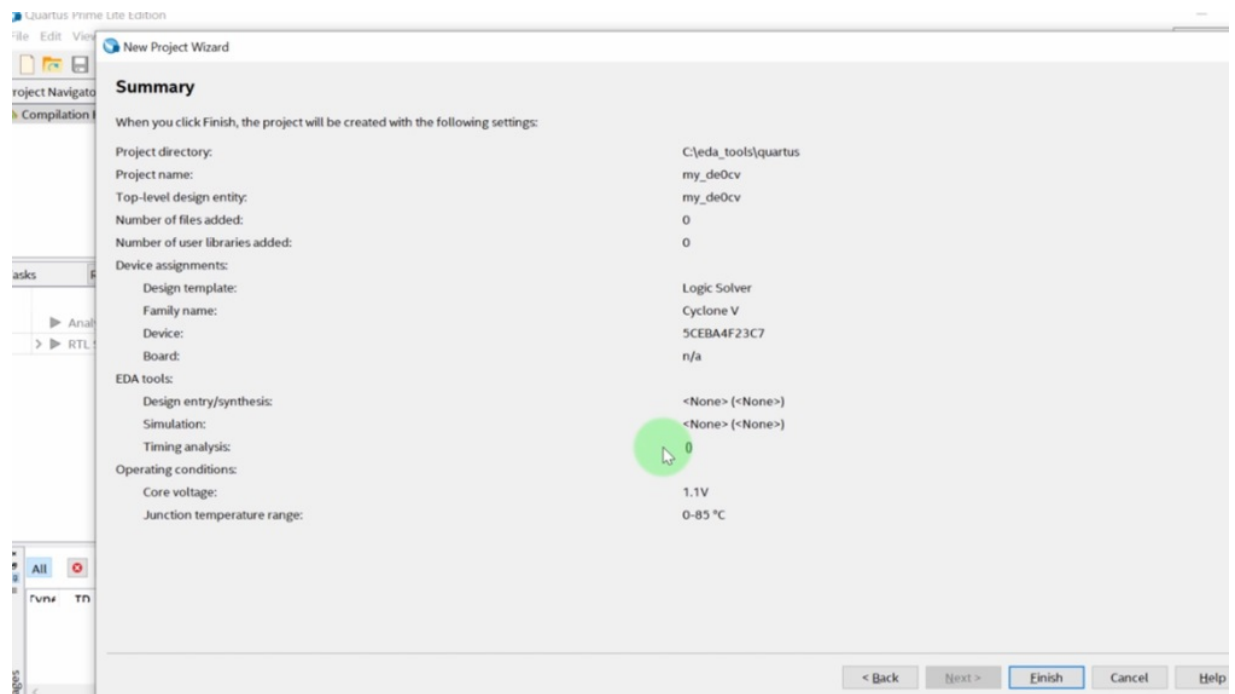


So let me choose from this list. The zero CV Development Board. And these are all of the available elements as you can see there are several versions here. And the difference is not the template version. They are all 1.0. The difference is the software version. And as you can see the first two elements are virtually the same. There are no differences in their specification.



So I can choose either of these two. This one says cyclone 5 the zero Development Board baseline burnout. So this has all of the spin assignments in the board for you to use on your top design. You could always use any of the others. And we even have an example design which is this logic solver. This is the one we saw online. So as you can see it's already installed. Let's go back to the browser and let's look at the version of this design so the design is also version 1.0. The device is five CB A4. That's the FPGA which is the same for all of these devices right here and the version of Cordis prime is eighteen point one. So what they are seeing here is that this version of the design is supported by our version of quietness. Once again there's no need to download it since we already have it here. Now that's not what it says right here. It says warning this elected the same template is not supported in the current version of the as spraying software download the latest version of the design template from the design store. So here are again the instructions to add design templates first download design templates from the design store second install the design templates. This is another link. And here you get to install them. You get a dot bar file. And here is where you will install it. That's the destination. Notice that the suggested

directory is the one for my project not the installation directory for quarters.

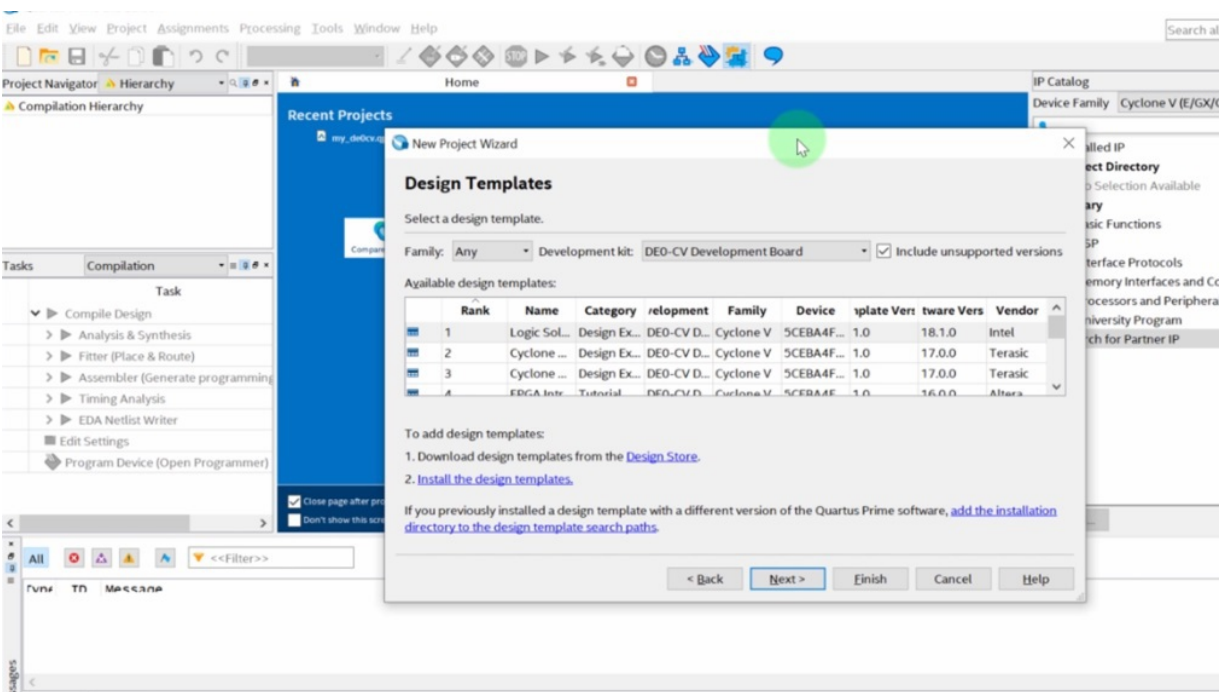


OK so let's go download that design and see how it looks in this list. Let's go back and download. We have to agree to some terms and we get to choose that file from courts. So here it is logic solver gooey duck bar. Let me click OK we get a message that the installation was successful and now the design should appear here. Let's select the D zero C.V. once more and here it is. Now it's the first one and you'll see that the software version is eighteen point one. Once again since the template version is 1 all we can understand by this is that this version was already supported but the download we just made informed the idea about this support. So I question how necessary that was. But for example if you have another board that is not listed in here or some example that is not in this installation you can always download it following these steps. That's why I've been showing you all of this. So let's click next and we get a summary so we get to click on finish after a while we get our design finally open in quarters. So at the left we get the hierarchy with the logic solver

shown right here. Remember this is an already made design so it has a bunch of code in very long and it's a nice application but it requires some explanation and I encourage you to go through the code and try to figure out what it does and maybe even download it to your DRM CV board if you have one. But for now let's see how we can make a design of our own.

HELLO WORLD - CREATE A PROJECT

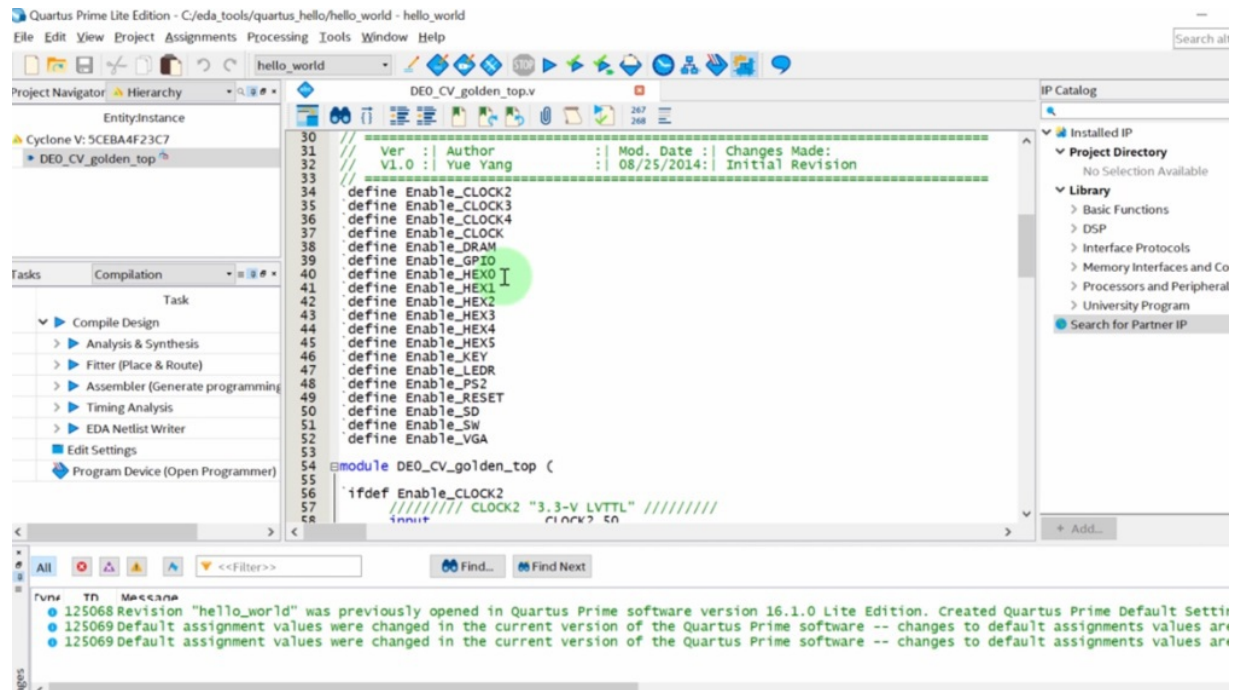
So now let's create a project of our own. And for that we need to click on new project wizard next. And a very important thing is that we are not supposed to work on the same directory for different projects. So I'll rename this the latest project was on a tool Skordas. So let me name this quietness.



Hello the project will be called Hello World. Next we'll create a project template again but this time we'll create an empty project for that the zero CV board. Well not exactly an empty project but the baseline keynote project once again we get the message that this is not supported by our version of quarters. But I say let's take our chances finish.

HELLO WORLD - TOP MODULE SOURCE FILE

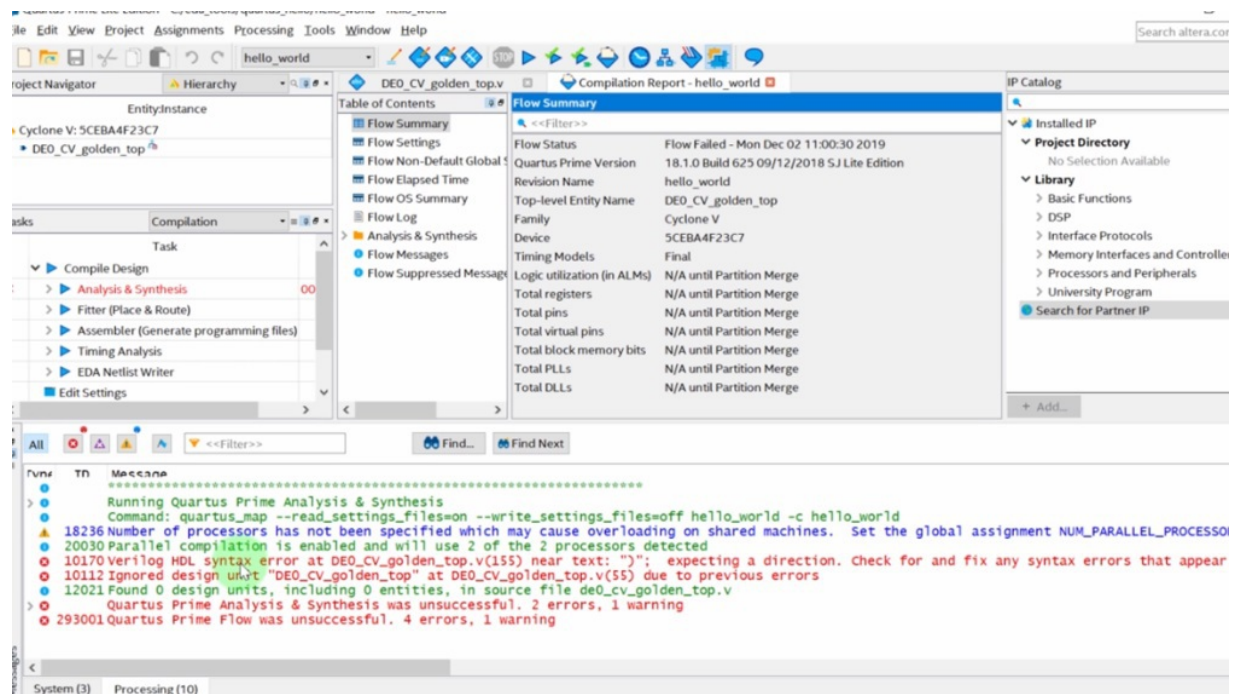
And in here we get one single design that says the zero CV golden top after opening it will see a big comment right here with the copyright and information about this project and then we have a bunch of defined statements which are very very useful they say define enable clock to clock three clock for enable every device on the board.



```
30 ///////////////////////////////////////////////////////////////////
31 // Ver    :| Author    :| Mod. Date :| Changes Made:
32 // V1.0  :| Yue Yang  :| 08/25/2014:| Initial Revision
33 ///////////////////////////////////////////////////////////////////
34 `define Enable_CLOCK2
35 `define Enable_CLOCK3
36 `define Enable_CLOCK4
37 `define Enable_CLOCK
38 `define Enable_DRAM
39 `define Enable_GPIO
40 `define Enable_HEX0
41 `define Enable_HEX1
42 `define Enable_HEX2
43 `define Enable_HEX3
44 `define Enable_HEX4
45 `define Enable_HEX5
46 `define Enable_KEY
47 `define Enable_LED
48 `define Enable_PS2
49 `define Enable_RESET
50 `define Enable_SD
51 `define Enable_SW
52 `define Enable_VGA
53
54 module DE0_CV_golden_top (
55
56     `ifdef Enable_CLOCK2
57         /////////////////////////////////////////////////// CLOCKS ///////////////////////////////////
58         input          CLOCKS2_50
```

Hex 0 x 1 which are the hex displays enabled key for the key arrays or the buttons. We have the switches right here. Enable as W. ala the R which are D LCD onboard. And if you leave all of these defined will all of these elements will be included in your design and the way

This is the compilation purpose of the project and in the compilation purpose is where we get to program the device in the end. So we are good. This is what we want. Remember that we could select to perform only a simulation in the gate level or the RTL level.



We can also go to full design which allows us to perform a lot of verification after we have downloaded our prototypes. But let's stick to compilation and you know what. In almost every EDTA too if you choose to perform the last operation you want all of the required operations and only the required operations will be performed. And so we can rest assured that we can simply generate the programming file by choosing this element right here. So we only have to double click on assembler. Let's do that we have to save our changes and of course this will take a while. You can always see the messages in this section below now. This is embarrassing. We got four errors let's take a look at the messages and we got a never in line one fifty five. It says syntax error near text closing parentheses so the syntax error is close to this 90 and I see the error remember how I told you that Intel or Altera should have tested this code. Well apparently they didn't. If you get to use the switches or any other

elements but not the V-J you would still get this comma here and the ports list is supposed to end without a comma.

0

The screenshot displays the Quartus Prime Lite Edition interface. The main window shows the 'Flow Summary' for the compilation of 'DE0_CV_golden_top.v'. The flow status is 'Successful - Mon Dec 02 11:04:44 2019'. The flow summary table lists various components and their utilization:

| Component | Value |
|---------------------------------|----------------------------------------------|
| Quartus Prime Version | 18.1.0 Build 625 09/12/2018 S J Lite Edition |
| Revision Name | hello_world |
| Top-level Entity Name | DE0_CV_golden_top |
| Family | Cyclone V |
| Device | 5CEBA4F23C7 |
| Timing Models | Final |
| Logic utilization (in ALMs) | 1 / 18,480 (< 1 %) |
| Total registers | 0 |
| Total pins | 20 / 224 (9 %) |
| Total virtual pins | 0 |
| Total block memory bits | 0 / 3,153,920 (0 %) |
| Total DSP Blocks | 0 / 66 (0 %) |
| Total HSSI RX PCSs | 0 |
| Total HSSI PMA RX Deserializers | 0 |
| Total HSSI TX PCSs | 0 |
| Total HSSI PMA TX Serializers | 0 |

The console window at the bottom shows the following output:

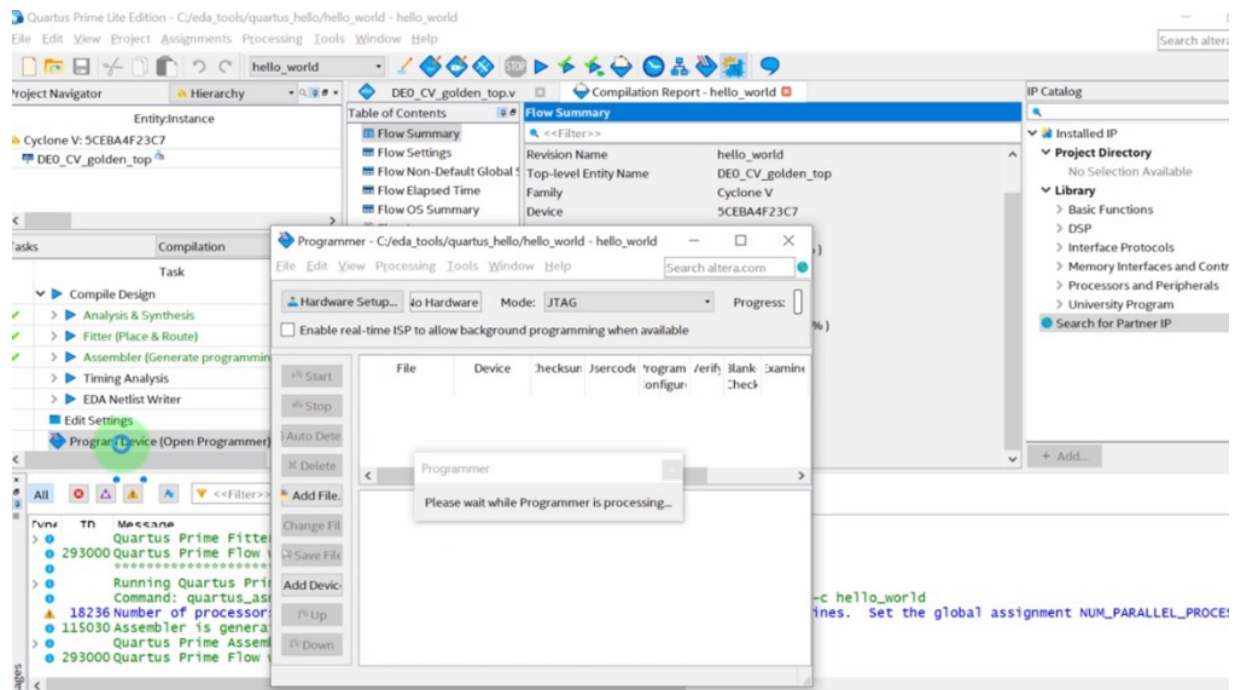
```
Time: 293000 Quartus Prime Fitter was successful. 0 errors, 195 warnings
293000 Quartus Prime Flow was successful. 0 errors, 196 warnings
*****
Running Quartus Prime Assembler
Command: quartus_asm --read_settings_files=off --write_settings_files=off hello_world -c hello_world
18236 Number of processors has not been specified which may cause overloading on shared machines. Set the global assignment NUM_PARALLEL_PROCESSOR
115030 Assembler is generating device programming files
Quartus Prime Assembler was successful. 0 errors, 1 warning
293000 Quartus Prime Flow was successful. 0 errors, 197 warnings
```

So that's a rookie mistake not made by me. But I know what I can do to fix it. I can simply delete this comma and let's try it again generate the programming files. Let's double click here. Let's save and hopefully wait for a while to get a success message. All right. So finally we have success right here. It's this quietest Brian flow was successful. It has zero errors but it has a lot of warnings and with as many as one hundred and ninety seven warnings I must at the very least urge you to take a look at them. Remember that a warning is an error waiting to happen anyway. That's keep on taking our chances. Please bear with me. So now we get some reports we have a table of contents here with all of those reports. So here's the flow summary. Remember flow means the whole tool chain. And by that right now we mean only analysis and synthesis the. Which is what Intel calls police and route and the assembler or the programming file generator. Basically we have synthesis implementation and programming file generation. The steps we recently became familiar with. And so here we have all of the results.

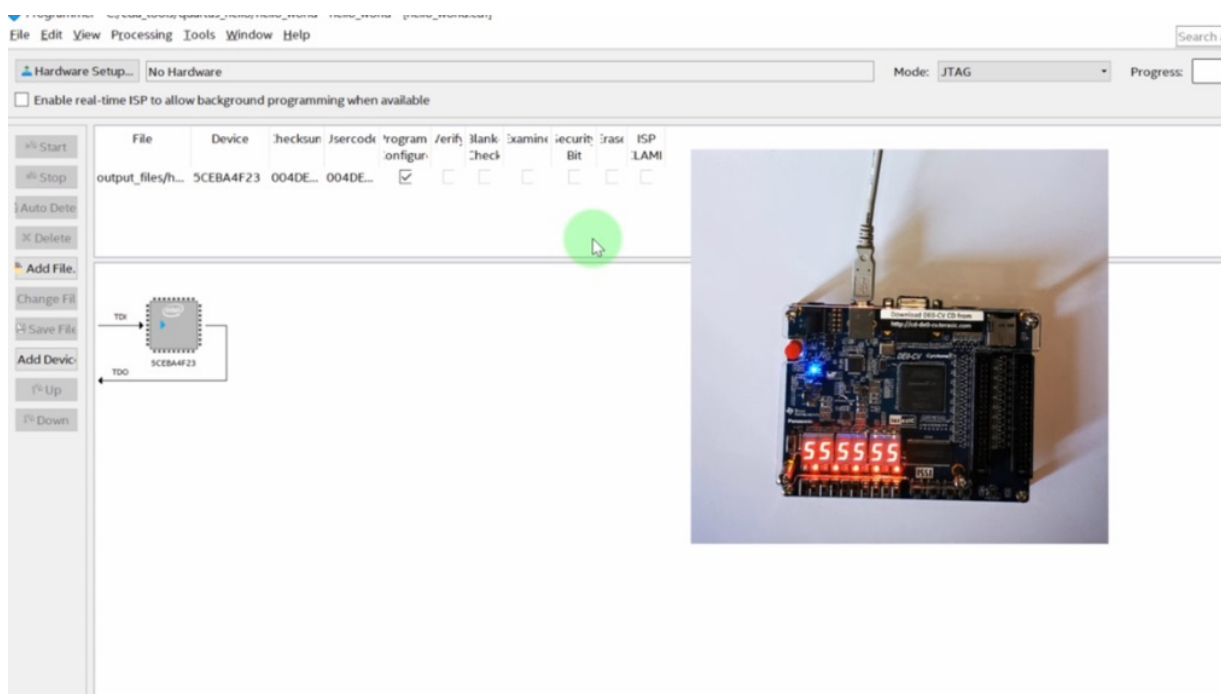
If you are interested in looking at them and you can look at a lot of stuff here which will become more and more relevant to you as you gain experience. So now all that's left is to download this design into your the zero CV board which is what we'll do. Up next.

HELLO WORLD - PROGRAMMING THE FPGA

So now let's get this design into our easier CV board. So all we have to do is go to the tasks section and scroll down to program device. This will open the program application so let's do that now in order to use this application we need to connect the board to the computer with a USB cable. So let me do that and noticed that the demo application is running with a count being sent to all of the displays and the LCD also blinking and if this is the first time you connect your board to your computer your operating system will ask you for permission to use it.



You may be asked to install some drivers but you must have installed the USP blaster already when you installed cordless so here we have the target device. And on top we have the information for the application we are going to download into it. For now the defaults will do which is only program and configure and at the left we have the tasks that we may perform. But notice that we cannot push this Start button because the hardware isn't recognized yet. So we have to click on hardware setup and we should be able to find our U.S. be blaster somewhere in this list.



Here it is. So we must use this dropdown and select the U.S. B blaster. That's it. Now we may close and we are good to go. Let me click on Start and as you can see at the right we get 100 percent. This was supposed to be the progress but it happened really fast. And it's this successful. So now if we pay attention to the board you'll see that the states in the switches are replicated by the LCD which is what I wrote in the very loud code. Now remember this is in REM. So if I cycle the power to the board you'll see that the demo application is running again.

EDA PLAYGROUND, BY DOULOS

Now it's time to talk about EDTA playground so let me tell you what we'll see in this project. First I'll tell you about a playground and its vendor Douglas. We'll talk about the required setup. I'll give you a quick walkthrough of PDA playground.

In this lesson: EDA Playground

- About EDA Playground and Doulos
- Setup
- A Quick Walkthrough
- Code Example
- Simulation

I'll show you a code example and some simulation. Like I said before even a playground does not support implementation so that's where we'll stop at simulation so let's get started.

ABOUT EDA PLAYGROUND AND DOULOS

So let me tell you about E.D. a playground and do loss in a playground is an online tool. That's why the logo has this big cloud right here and being an online tool has a lot of implications. First you don't have to install anything. Second you get to run on many many many platforms because you have several operating systems and several web browsers. So it's a very convenient tool. If you've been taking the whole series you may remember that this tool is featured in our first project the very low HDL project and we saw how to simulate on a playground. Again in this project I'll show you some of the tricks we saw in the first project but hopefully you'll get to learn many more tricks and even reinforce what you learned in the first project. A very cool feature is that it's free.

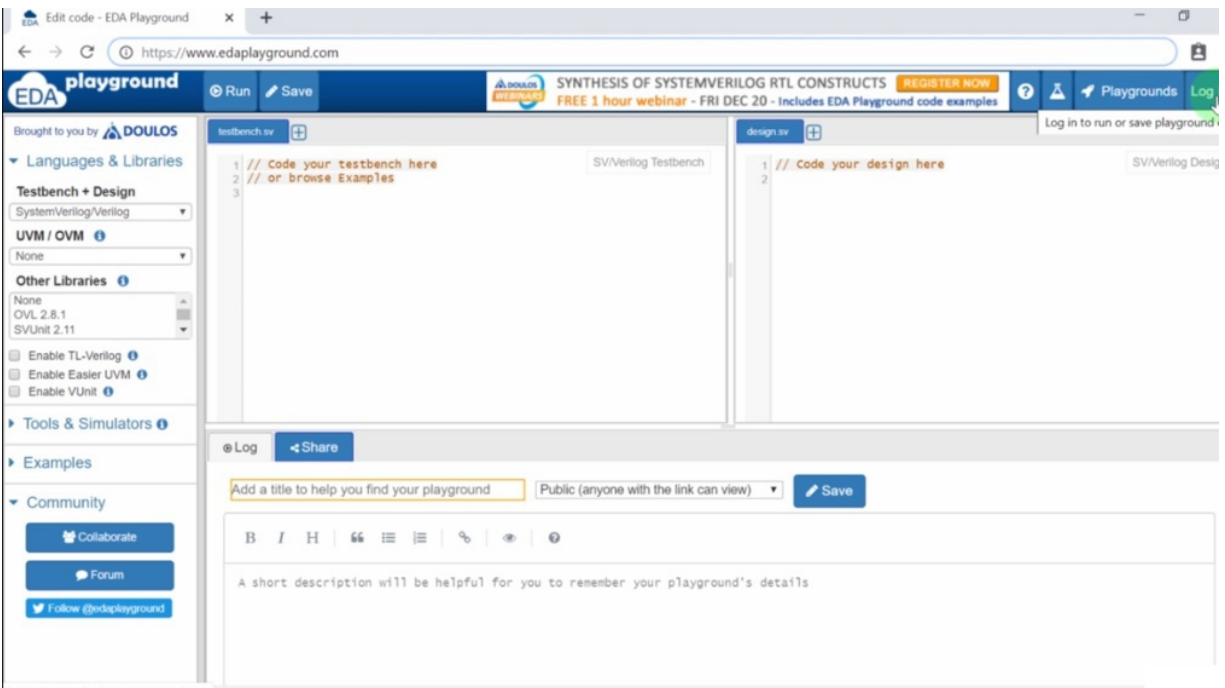
About EDA Playground and Doulos

- EDA playground is an online tool.
- Featured in our first course: **Verilog HDL**.
- It's free!
- You simply need to register.
- It supports several languages and tools.
- Made by Doulos, a company specialized in training.

You don't need to pay anything. All you have to do is register. You can do that with Google or Facebook or with your personal e-mail account one of the coolest features of PDA playground is that it supports a lot of languages and tools it supports very low the HDL system very low. And I've seen some other languages there. As for simulators it supports a lot of simulators. The one I like the most is Icarus very long because of two reasons. First it's free and second. Well I love very low and speaking about free tools. There are commercial tools like simulators in a playground but you don't have to pay for them. You simply have to verify your account. I'll tell you how later. And finally the playground was made by Douglas. This is a company that specializes in trainings. So you may want to check their Web site which is to list dot com and you may learn a lot for free from them. This is a very very cool company.

EDA PLAYGROUND SETUP

So here we have either a playground as you can see the U.R.L. is either a playground dot com. Now let me show you the requirements for this software so I'll show you this setup. You need to perform in order to use this application needless to say this is an online tool so you only need to have a good browser and a good operating system. The requirements are pretty much what you would use with any other online tool. Nothing fancy.



So Google Chrome or Firefox will do in either Mac OS or windows or pretty much any distribution of Linux so that's for the technical requirements but to use this tool you'll need to create a user. So let's go here at the top right. And here's the log in button. So let's press

this one and here you get a screen where you can choose which type of user you want to create. So you get two options. You can log in with your social accounts. That's Google or Facebook or you can create your own user to I.D. a playground.

EDA playground

SYNTHESIS OF SYSTEMVERILOG RTL CONSTRUCTS REGISTER NOW
FREE 1 hour webinar - FRI DEC 20 - Includes EDA Playground code examples

Log in with one of the following providers:

log in with Google log in with facebook

Logging in with a social accounts gives you access to all non-commercial simulators and some commercial simulators. If you want to use all the commercial simulators, please register for an account below.

No Google or Facebook account? [Privacy Policy](#)

or

Want full access to EDA Playground?

Username
Password
Login

[Register for a full account](#) [Forgotten password](#)

To run commercial simulators, you need to register and log in with a username and password. Registration is free, and only pre-approved email's will have access to the commercial simulators.

If you wish to use EDA Playground as a playground, please log in using your Google or Facebook account via the links above.

Now here's a nice note that says if you wish to use a playground as a playground please log in using your Google or Facebook account via the links above. So this is the recommended type of user. But with this user with this regular user you'll get to use only non-commercial simulators like Icarus very like for example. This is the one that I'm using in this project. And that's the one I usually use. But if you want to use a commercial Simulator you'll have to register fully as a full access user. And this only requires a second step of verifying your e-mail address. This registration is still free. So there's no additional effort on doing this anyway. Let me log in with my Google account and once you've created your account or once you've logged in this is the screen you'll get which is the EPA tool itself so to use this tool you can simply start editing the code and you can save your designs right here at the bottom. And this is what you'll learn how to do. Up next.

WALKTHROUGH_CODE ENTRY

So let me give you a walk through of the elements an idea a playground. So the first thing I'll do is show you where everything is. So like I said First we have the added windows right here. We have to edit windows because the one at the left is the test bench module you will use. And it has the name test bench. That is v for system very long. And at the right we have your design your top design or your many designs. Actually this one is called Design dot as V and you cannot change these two names. You can however change the names of the other modules you enter. So for example to enter a new design you can enter it here with this plus sign and you can upload your files or you can create a file. Let's create one named my gate. So here it is.

The screenshot shows a web-based Verilog playground interface. At the top, there is a navigation bar with a 'Run' button, a 'Save*' button, and a banner for a 'SYNTHESIS OF SYSTEMVERILOG RTL CONSTRUCTS' webinar. Below the navigation bar, there are two code editors. The left editor, titled 'testbench.sv', contains the following code:

```
1 // Code your testbench here
2 // or browse Examples
3 module my_device_TB();
4   reg x,y,z;
5   wire my_out;
6   my_device Dd1_my_device(x,y,z,my_out);
7
8   initial begin
9     en1;
10
11 endmodule
```

The right editor, titled 'design.sv', contains the following code:

```
1 // Code your design here
2 module my_device(input a, input b, input c, output x);
3   wire temp;
4   mygate mg1(a,b,temp);
5   mygate mg2(temp,c,x);
6 endmodule
```

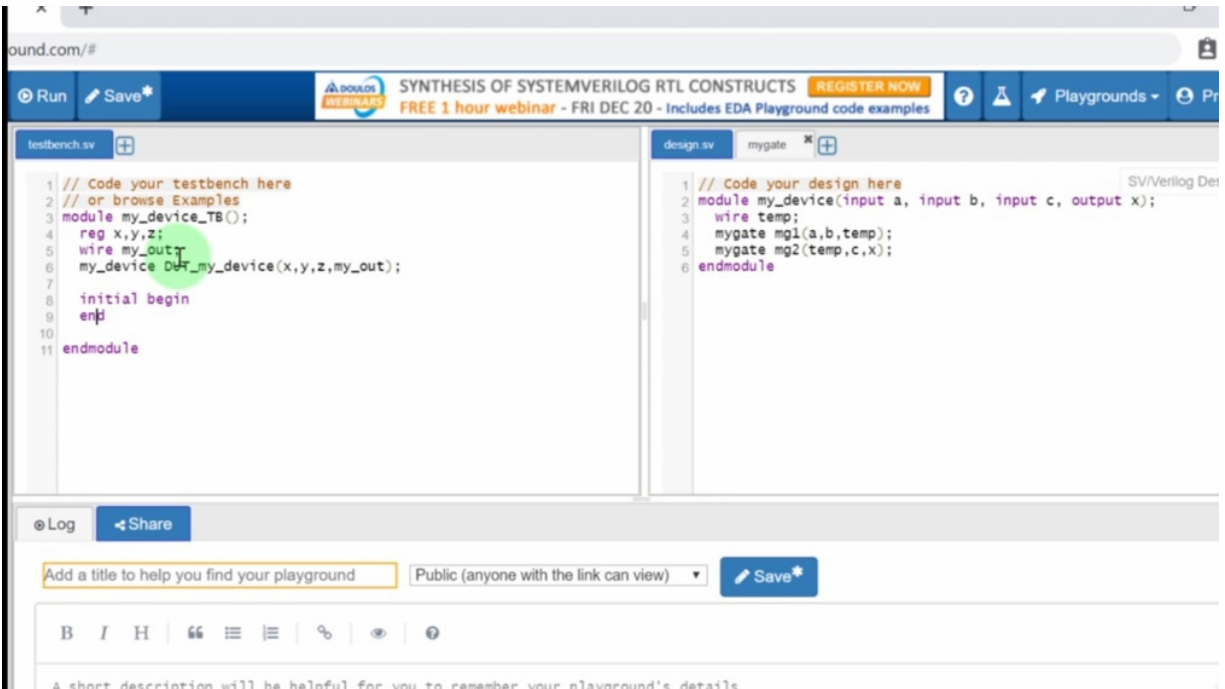
Below the code editors, there is a 'Log' button and a 'Share' button. A text input field is labeled 'Add a title to help you find your playground' and a dropdown menu is set to 'Public (anyone with the link can view)'. There is also a 'Save*' button. At the bottom, there is a rich text editor with formatting options (B, I, H, bold, italic, underline, link, unlink, list, list, link, unlink) and a placeholder text: 'A short description will be helpful for you to remember your playground's details'.

Let me just write this new module. My gate which has an input a an input B and an output C and let's say this will be a NAND gate Nothing fancy here let's say that my top design uses that NAND gate for example my device lets say that it has three inputs A B and C and when output X and let's say this uses to cascaded NAND gates. Notice that the result will not be the NAND operation between A B and C but rather the composition of two NAND gates so I'll use one my gate name energy one which takes in A and B and outputs a wire named temp and my other gate will be called MDG 2 which takes in temp and C and outputs X so that's it. That's a top design I'm just doing this for you to get familiar with code entry. Now at the left I can make my own test bench so I'll make a module my device D.B. you know I like to keep some naming convention for test benches and I usually named them as the module I'm testing underscore DP for test bench and then right here let me just instantiate one my device module so it will be called duty for device under test under score my device let me declare some stimulus variables so these will be registers X Y and Z and I'll use a wire for the output let's name it my out so my device under test will receive X Y and Z and it will output my out. Now let me set the simulation right here and I will stop the simulation with the stop task and let me use just one task that says display Hi there.

WALKTHROUGH_LEFT TOOLBAR

Let's run the simulation. And for this I will be prompted to select a valid tool or simulator. So that's where we have the whole flow of development at the left. So here we have languages and libraries. We also have another section for tools and simulators. We have examples we could use and we have community resources. So looking at languages notice the system very low and very low is already selected but we could use some other languages even program in languages like perl or C.. But here we have system very low and the HDL will stick to very low. These are verification languages we are not using these. So we will not select any. We could use libraries also but we are not using them. So I'll collapse this and I will expand tools and simulators. The simulator I usually recommend is Icarus very low. And the version I like is the latest zero point ten. Here you get to enter some options for the command line and notice that if I tried to run this I may get an error we may save our design and if you want to save your designs you get to enter a name and specification for it. So if we hit save this will be saved without a name. And it's always a good practice to name your projects. So this is where you get to do that. I'll name this one walkthrough through of FDA playground for the description. I will just enter a walk through a PDA playground for the FPGA and better designed project series. And you get to set the visibility of your project right here. You get to make it public so that anyone can view it but you can make it published so that it appears in search results. So the published visibility is more accessible than the public one and you can make it private so that only you can see it. Anyway we'll leave this as published so you can look for it if you want. Let me press save and now we get to run our simulation because we have selected a tool. So let me hit run and we get an error. It's this syntax

error in line eleven of the test bench. And as you can see line eleven is the end keyword. But previously my stop desk didn't have a semicolon. So that's a rookie mistake. Let's run once more and this is the error. I told you about so we get that in lines four and five of these signed up as V right here. Lines four and five have an unknown module type. That's my gate but my gate is already defined here. So you are supposed to tell the truth Jane about all of the source files you are using. So you get to put this in the compile and run options which are the command line options. So let me just enter that in this line it says w all and G 2012. That's the version of the compiler and first let me change the name of this file. My gate I forgot to put an extension to it. I'll name it my gate that V. Not that it matters. You may name your files as ever you want but to include them in your project. You have to add their names after all of the options you have entered here. So let me type in just space my gate that V. If I had any more source files I would just add them here separated by spaces for example more dot the and now let me just click Run and there it is. It did compile it synthesized All right and it sent the high their message to the output right here. So here's the output I sent. And as you know I may run a simulation here changing the stimulus variables x y and z to see how my device works. You may want to do it as an experiment.



Now notice that the stop task didn't actually stop my simulation. I have to click here on stop and now notice how the finish task works let me hit run and notice that it says Hi there. And finally done so the simulation did stop. So there's a difference between finish and stop. I'll tell you about it a bit later so moving on under tools we get to use E.P. wave which is the way from viewer from a playground and I recommend that you use the way from viewer in order to see what's going on. You can always report through display or monitor to the console output right here. So you have many options. You can also download your files after running by clicking here. Next let's look at the examples. So here they are. We have examples for VHS Yale very low end system very low and all of the languages that are supported I invite you to browse and take a look at all of these examples. You will learn a lot from these examples. And in the community section you get to collaborate. You get to join forums and you get to follow the playground on Twitter if you want. Speaking of which in this shared tab you get to not only save your files but you get to work with likes. The description has rich tech support with links and other cool features and you can share your designs via Twitter Facebook LinkedIn or with the link.

WALKTHROUGH_OTHER TOOLS

Moving on with the walkthrough let me just save and show you that you can always get help from the documentation with this question mark sign. Let's click on it. So here's the whole documentation of a playground. So if you feel something's lacking from this project you may always come here. Remember Douglas is a training company so they have done an excellent job at providing free trainings for their tools going back. We also get to see which apps are available at the moment. And as you can see we have a playground and AP wave. It is a playground it's just a link back to a PDA playground and AP wave is well the way forum viewer.

The screenshot displays the EDA Playground web interface. At the top, there are browser tabs for 'Walkthrough of EDA Playground' and 'EDA Playground Documentation'. The address bar shows 'edaplayground.com/x/5QKu#'. The main interface features a blue header with the 'EDA playground' logo, navigation buttons for 'Run', 'Save', and 'Copy', and a banner for a 'SYNTHESIS OF SYSTEMVERILOG RTL CONSTRUCTS' webinar. On the left, a sidebar lists 'Languages & Libraries', 'Tools & Simulators' (including Icarus Verilog), 'Compile & Run Options', 'Examples', and 'Community' with buttons for 'Collaborate', 'Forum', and 'Follow @edaplayground'. The central workspace is split into two code editors: 'testbench.sv' containing a Verilog testbench and 'design.sv' containing a Verilog module definition. Below the code editors, there are social media sharing options (Log, Share, Twitter, Facebook, LinkedIn) and a text area with the title 'Walkthrough of EDA Playground' and a description 'A walkthrough of EDA Playground for the FPGA Embedded Design Course Series.'

Now this is an example wave that is being shown right here. I don't even know which project it belongs to but anyway let's go back. And here we have at the playground dropdown that you may browse for your playgrounds or the published playgrounds. Let's take a look at your playgrounds which are the ones made by my user. And you can browse everything you've done here so I have all of the examples I made for the very luck project available here. You may have already seen these and at the very top you can see the walk through of a playground. I just created no notice.

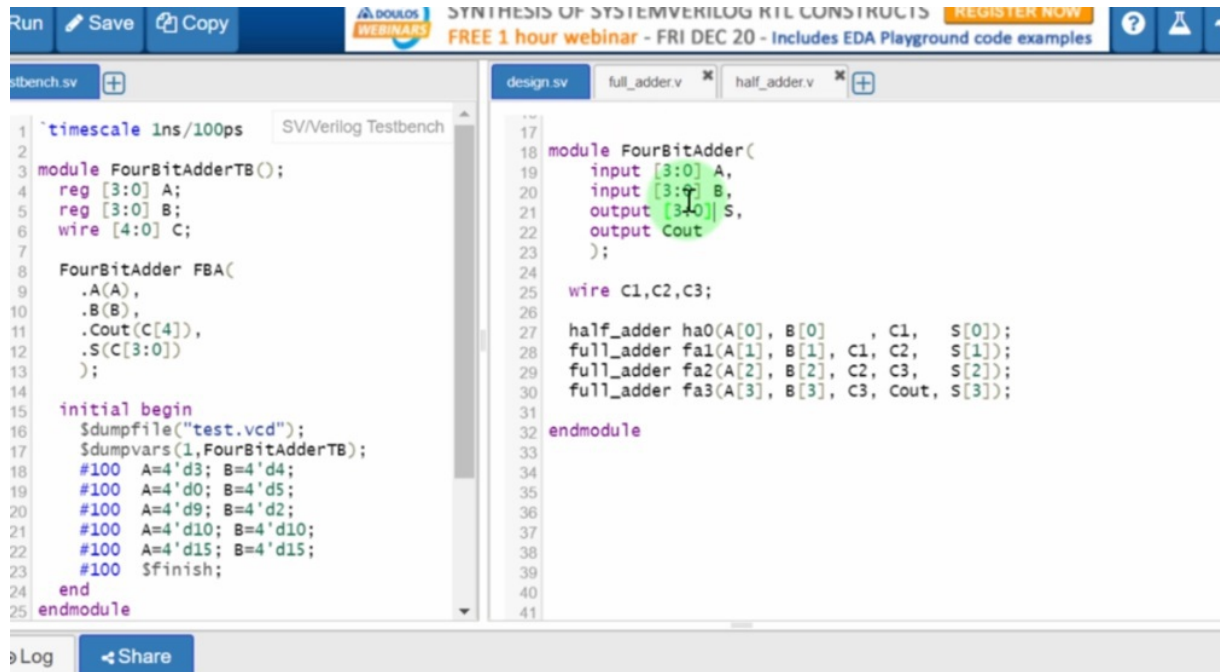
| Name | Description | Public? | Modified | Likes | Views |
|-------------------------------|--------------------------------------------------------------------------------------------------------|-----------|--------------------|-------|-------|
| Walkthrough of EDA Playground | A Walkthrough of EDA Playground for the FPGA Embedded Design Course Series. | Published | 2019/11/26 12:28pm | 0 | 0 |
| Constants | | Private | 2019/10/28 8:33pm | 0 | 0 |
| My 4 digit clock | | Public | 2018/11/01 2:00am | 0 | 4 |
| State Machine | | Public | 2018/10/16 8:51pm | 0 | 0 |
| Tristate | | Public | 2018/10/16 6:35pm | 0 | 0 |
| pruebas del curso de verilog | | Public | 2018/06/11 10:57pm | 0 | 0 |
| Multiplexer by assign | # **assign example** This example shows how to implement a 4 to 1 multiplexer by means of an assign... | Public | 2018/03/09 10:46pm | 0 | 0 |
| Demultiplexer by if-else | # **if-else example** This example shows how to implement a 4 output | Public | 2018/03/09 10:12pm | 0 | 0 |

The visibility I've set for my project. Most of them are just public. You can look for them under my user and there is just one that's published. I wonder if I should make the other ones published and notice that you get a very nice search engine here not specially to look for your own projects but for the published projects right here. Published playground so let me search for a project from this whole bunch let's look for the one I just published walk through here it is. Walk through of a playground user me and noticed that if we look for a project of mine that is public for example Multiplex or buy a sign let's look for it in the published projects it will not show so that

answers my question yes I'm supposed to make all of my projects published so that you can look for them. As of this recording you will find them all published and finally we have the profile dropdown here where you get to set up your profile or log out. Let's see my profile. Here you can see your playgrounds the published playgrounds which are the two pages we just saw and your user profile. Now here's a nice option I like to check. This is open E.P. wave way forms on a separate page after run. That is because when you run your applications with the way form viewer your browser will show you AP wave instead of PDA playground so you won't get to see the code and your way forms so that's it for the walkthrough. Now let's take a look at an example design.

DEMO_CODE EXAMPLE

So here's the code example I chose. This is another name for the ripple carry. So here at the right. Let me show you the source code so my design is quite simple. It is a four-bit adder. It has two inputs A and B and an output S with a carry output. Both the inputs and the output are four bits. So I have three intermediate wires for the carry output one, two, and three.

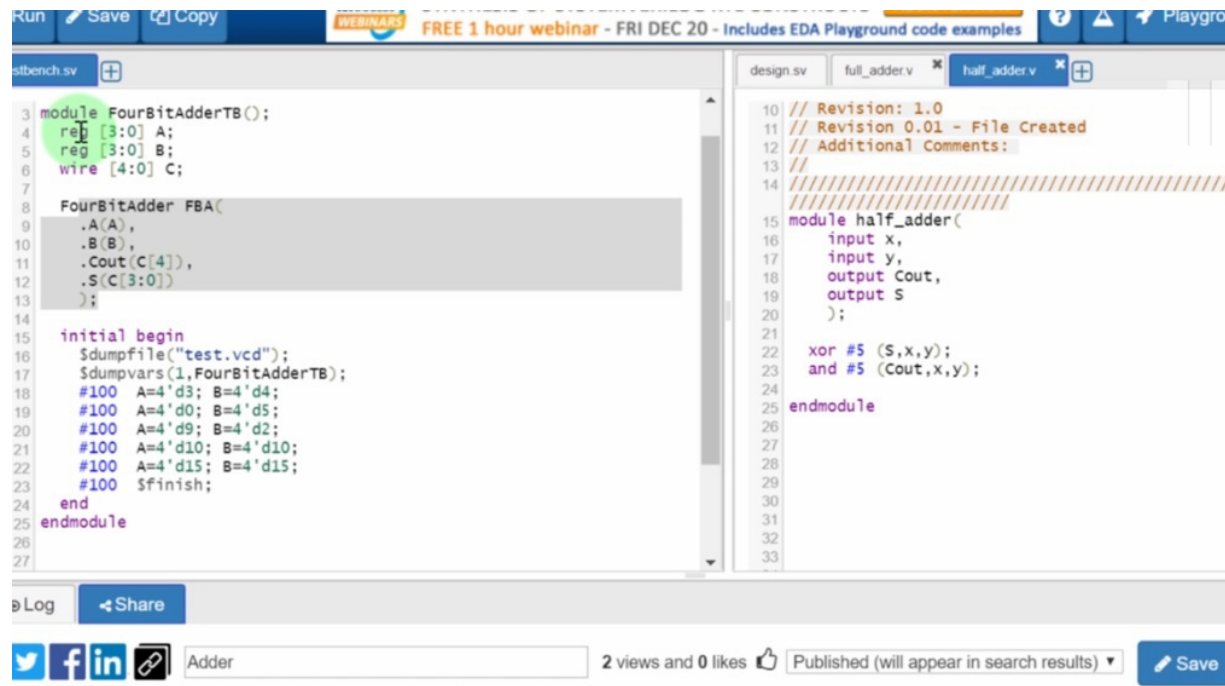


```
1 `timescale 1ns/100ps
2
3 module FourBitAdderTB();
4   reg [3:0] A;
5   reg [3:0] B;
6   wire [4:0] C;
7
8   FourBitAdder FBA(
9     .A(A),
10    .B(B),
11    .Cout(C[4]),
12    .S(C[3:0])
13  );
14
15  initial begin
16    $dumpfile("test.vcd");
17    $dumpvars(1,FourBitAdderTB);
18    #100 A=4'd3; B=4'd4;
19    #100 A=4'd0; B=4'd5;
20    #100 A=4'd9; B=4'd2;
21    #100 A=4'd10; B=4'd10;
22    #100 A=4'd15; B=4'd15;
23    #100 $finish;
24  end
25 endmodule

17
18 module FourBitAdder(
19   input [3:0] A,
20   input [3:0] B,
21   output [3:0] S,
22   output Cout
23 );
24
25 wire c1,c2,c3;
26
27 half_adder ha0(A[0], B[0], c1, S[0]);
28 full_adder fa1(A[1], B[1], c1, c2, S[1]);
29 full_adder fa2(A[2], B[2], c2, c3, S[2]);
30 full_adder fa3(A[3], B[3], c3, Cout, S[3]);
31
32 endmodule
```

And so I'm using a half adder for these significant bits of A and B leaving our least significant bit of the output S and I'm using the carry output c1. Notice that the half adder only has four pins in it and then we have full adders which have five pins or ports in them. They have the carry input so the carry output of each bit is the carry input of the next one. That's exactly what a ripple carry does anyway. This

adds to four big numbers and outputs a five big number or a four big number with a carry output.



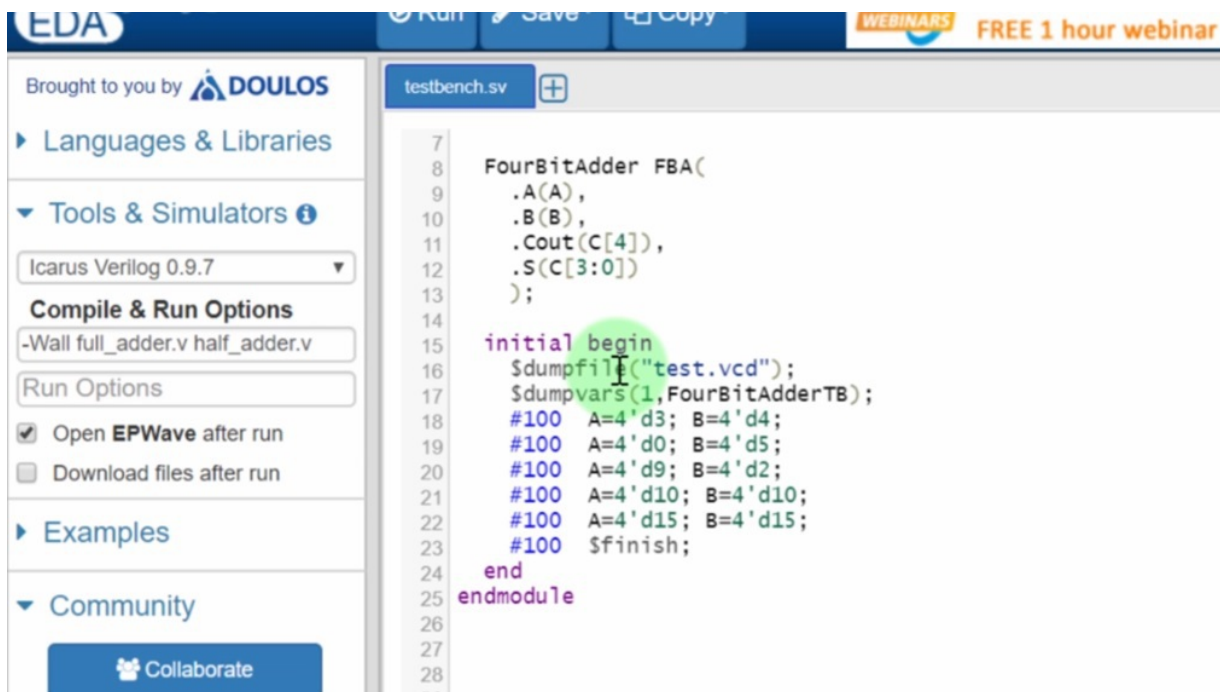
```
3 module FourBitAdderTB ();
4   reg [3:0] A;
5   reg [3:0] B;
6   wire [4:0] C;
7
8   FourBitAdder FBA(
9     .A(A),
10    .B(B),
11    .Cout(C[4]),
12    .S(C[3:0])
13  );
14
15  initial begin
16    $dumpfile("test.vcd");
17    $dumpvars(1,FourBitAdderTB);
18    #100 A=4'd3; B=4'd4;
19    #100 A=4'd0; B=4'd5;
20    #100 A=4'd9; B=4'd2;
21    #100 A=4'd10; B=4'd10;
22    #100 A=4'd15; B=4'd15;
23    #100 $finish;
24  end
25 endmodule
26
27
```

```
10 // Revision: 1.0
11 // Revision 0.01 - File Created
12 // Additional Comments:
13 //
14 ///////////////////////////////////////////////////////////////////
15 module half_adder(
16   input x,
17   input y,
18   output Cout,
19   output S
20 );
21
22   xor #5 (S,x,y);
23   and #5 (Cout,x,y);
24
25 endmodule
26
27
28
29
30
31
32
33
```

So I'm using a half hour and three full others. So here's my full other dot v file which uses the very well known for other THAT USES TO HAVE OTHERS AND THEN WE HAVE MY have other right here which uses only an X or an AND gate now my test bench module has one for a bit other instantiated with the stimulus variables here and the output C which has 5 bits in it. So when I use stanza 8 I use the Kerry output as C4 and the sum as C from 3 to 0. Anyway I'm trying to add three plus four then zero plus five then nine plus two then 10 plus 10 and finally 15 plus 15 I end this simulation with finish.

DEMO_SIMULATION DUMP SYSTEM TASKS

Now notice the dump file and dump virus tasks right here. These tasks are used because either a playground uses a synthesis tool that provides an output this output for the simulator.



```
7
8   FourBitAdder FBA(
9     .A(A),
10    .B(B),
11    .Cout(C[4]),
12    .S(C[3:0])
13  );
14
15  initial begin
16    $dumpfile("test.vcd");
17    $dumpvars(1, FourBitAdderTB);
18    #100 A=4'd3; B=4'd4;
19    #100 A=4'd0; B=4'd5;
20    #100 A=4'd9; B=4'd2;
21    #100 A=4'd10; B=4'd10;
22    #100 A=4'd15; B=4'd15;
23    #100 $finish;
24  end
25 endmodule
26
27
28
```

Since this is modular it may use one of many simulators the synthesis and. Well actually this simulation has to finish in order to provide the test that VCR the file. This file contains the gathered information for the simulation and this file is used by ERP wave which I will use to I must check here and EPA will take this file to show us all of the way forms the dump virus task allows us to gather information or to specify which variables we are going to dump into this file the first number is the level of depth we want to use. We are

fine with level 1 but we could use any level between 0 and 2 if you want to learn more about these tasks. You may make of course a web search and you will see how to use it. For example here so here is dump file where you specify the file and here Stump pass where you specify the level of detail.

```
$dumpvars(0, toptestbench_module);
```

When level is set to 0, and only the module name is specified, it dumps ALL the variables of that module and all the variables in ALL lower level modules instantiated by this top module. If any module is not instantiated by this top module, then its variable will not be covered.

. If you wish to also cover variables of a module that is not instantiated by the top module , you could write

```
$dumpvars(0, toptestbench_module, not_instantiated_module);
```

If we wish to dump the variables only in the top module but not the modules instantiated below it, we could have 1 as its first argument as in

```
$dumpvars(1, toptestbench_module);
```

This is helpful when you do not wish to be bothered about the registers and the variables inside the instantiated module. But, during the debug and to find root cause, it may be helpful to use 0 as its first argument and dump all variables.

The following example will dump all variables in the in the module named toptestbench_module and the modules one level below it.

```
$dumpvars(2, toptestbench_module);
```

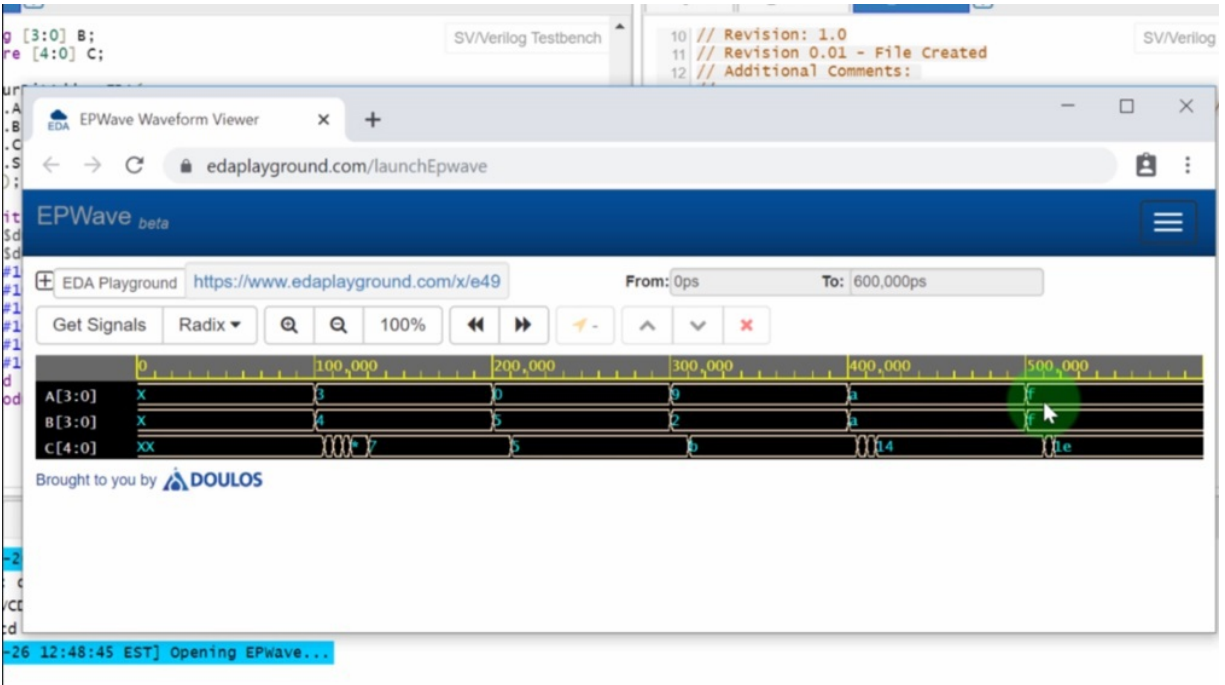
value of level more than 2 are rarely used.

Limiting the size of dump file

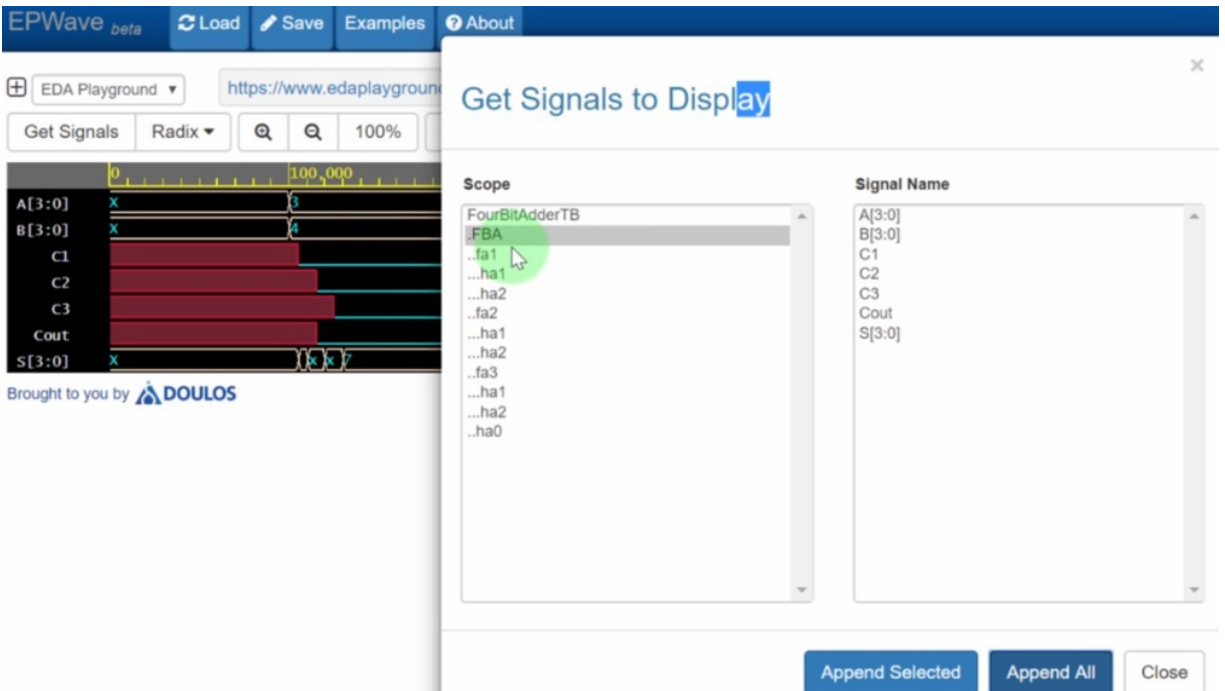
It is possible that you inadvertently generate huge file in Gigabytes (for examples while dumping a Gigahertz clock for one second). To reduce such occurrences, we may use \$dumplimit. It usage is

```
$dumplimit(<filesize>);
```

Now for this level of detail you have three options 0 1 and 2 so Level 0 includes all variables as you can see in this text with Level 0 you will see the variables from the top level design and the instantiated modules so you get to specify 0 and the top module. If you only want to show what's on the top level you will do what we have on the code which is Dunbar's 1 and the name of the module you are running and level 2 shows all of the variables in the specify modules and one level below it. So let's go back to our code. And as you can see we have Level 1. So this will only show a b and c that sold you'll see here if we wanted to look at the signals inside every full either and the half others we could use level 0. Let's see the simulation first for level 1.



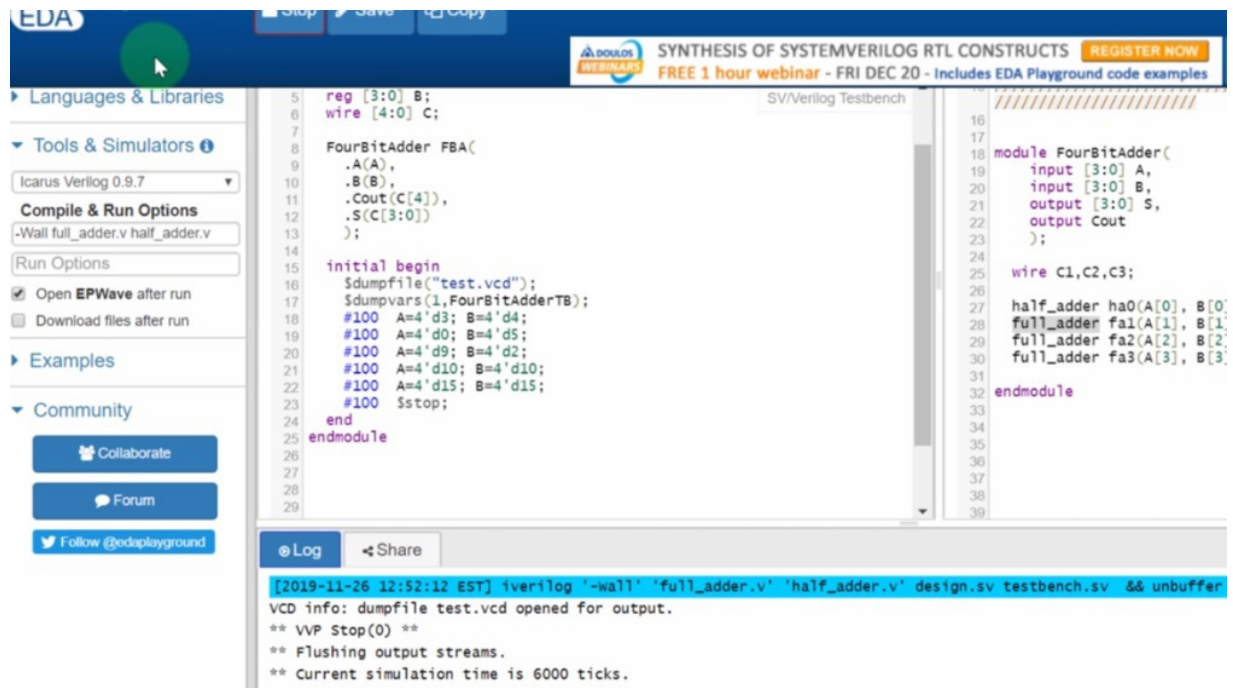
So here's the simulation. Like I said it would show in a separate window. Let me just resize and as you can see here we have three plus four equals seven zero plus five equals five nine plus two equals B A plus eight equals fourteen f plus F equals one. E if you want to see this in decimal you can't but you can see it in binary. If you want. Now notice the delays here. These are happening because my source code specifies delays in the gates. That was part of the main purpose of this simulation.



Now let me show you what happens when we change that level. Let's put Level 0 here. This will show every level. Let's run again and we are prompted to use the signals from a whole list that we have. So we have to use the get signals button. Here's to get signals button and here's the whole directory of signals that are available. If a one has all of these signals inside. If a one we have half either 1 and 2 if a to have other 1 and 2 and so 1. So let's say we select all of them all of the signals that are shown for FBA and then let's say we want to use all of the signals that are in each module that is instantiated you'll see that we can make a very complex plot if we want.

DEMO_SIMULATION CONTROL SYSTEM TASKS

And finally let me show you what happens if we use instead of finish here. The other simulation control task which is stop that's it. Run and notice that the simulation is running. It hasn't stopped. So we get to stop it with a button and we won't get the signals from E.P. wave until we stop this simulation. But if we hit stop right here we are telling a playground to stop the simulation altogether so we won't get any plots.



The screenshot shows the EDA Playground interface. At the top, there's a blue header with 'EDA' and a 'Stop' button. Below the header, there's a navigation menu on the left with sections like 'Languages & Libraries', 'Tools & Simulators' (showing 'Icarus Verilog 0.9.7'), 'Examples', and 'Community'. The main area is split into two panes. The left pane shows Verilog code for a testbench and a module named 'FourBitAdder'. The right pane shows the module definition for 'FourBitAdder'. At the bottom, there's a 'Log' section with a 'Share' button and a log entry: '[2019-11-26 12:52:12 EST] iverilog '-wall' 'full_adder.v' 'half_adder.v' design.sv testbench.sv && unbuffer VCD info: dumpfile test.vcd opened for output. ** VVP Stop(0) ** ** Flushing output streams. ** Current simulation time is 6000 ticks.'

So the difference between stop and finish is that stop holds the simulation without stopping the simulator and this tool chain requires

the simulator to output this file. So long story short in E.D. a playground if you want to use E.P. wave you have to stop your simulations not with the stop task but with the finished task. This is different for V or models in which is part of COURTIS And it's very important that you know the difference between Finnish and stop so let's see it once again. It's run with the Finnish task and there you have it.

VIVADO DESIGN SUITE, BY XILINX

Now it's time to talk about five other design sweet. Let me tell you what we'll see in this project. First I'll tell you about the value and sailings.

In this lesson: Vivado Design Suite

- About Vivado and Xilinx
- Setup
- A Quick Walkthrough
- Code Example
- Simulation
- Implementation

Then I'll tell you about the setup you need to perform. I'll show you again a quick walkthrough of this software. I'll show you a code example a simulation example and an implementation example just like quarters. Value is a full fledged to.

ABOUT VIVADO DESIGN SUITE AND XILINX

So let me tell you about the vato and sailings. First the Vive other Design Suite is a full fledged FPGA development tool. Once more here you'll see schematic diagrams. You'll see the floor plan inside the FPGA where the hardware goes you'll see pin assignments timing constraints the works. So it offers a free license. That's called the [REMOVED]. And so you get to download the vato HLS editions which are the ones currently available.

About Vivado and Xilinx

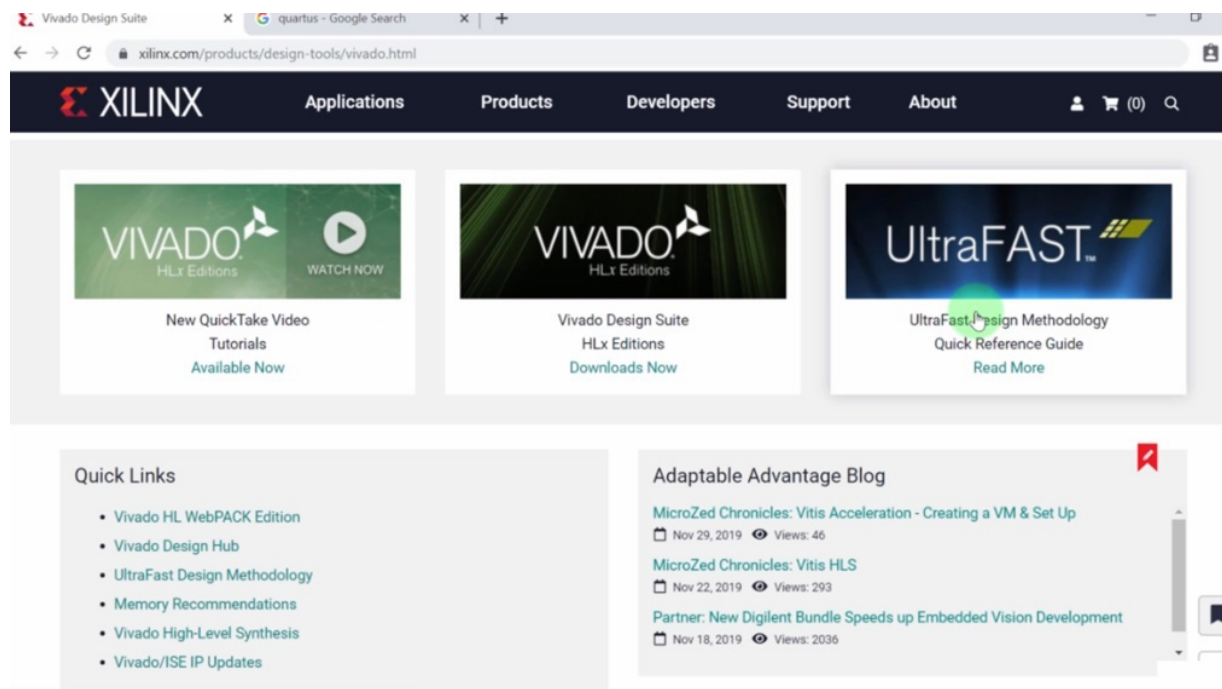
- Vivado Design Suite is a full-fledged FPGA development tool.
- It offers a free license called the *WebPack*.
- Setup starts at download.
- Made by Xilinx, the inventor of FPGAs.

And when you make your download you have to look for the web back license. It's a whole distribution for makers and hobbyists. That is free of charge just as with quarters setup starts at Download. So the installation tool asks you which elements you want to install and

you get to choose from a list. Again this is done to speed up download and installation this software was made by sailings and a nice detail about Xilinx is that they were the inventors of FPGA. At least that's what sailing exclaims. And I believe them.

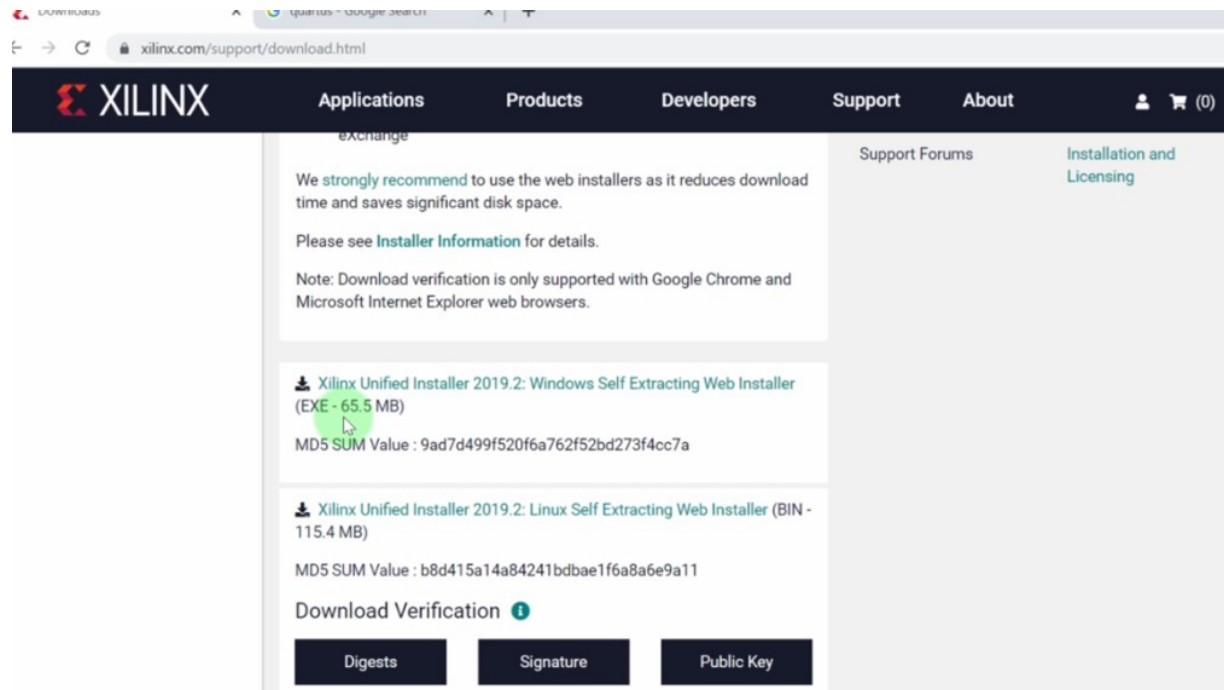
SETUP_ DOWNLOADING VIVADO

So the first step in working with sailings vibrato is of course downloading it so you can perform a web search. I've just looked for vato and the first result is Vive other Design Suite by side links. So this is give others Web site and I'm sure you can find the downloads just scrolling down.



Here are the different versions available and the one we are interested in is the version that contains the web back license. And this is called Duval HL Web edition. You can find it in several places in this page. I'll just click here so here you can find several old specifications of the whipped back license and the other licenses.

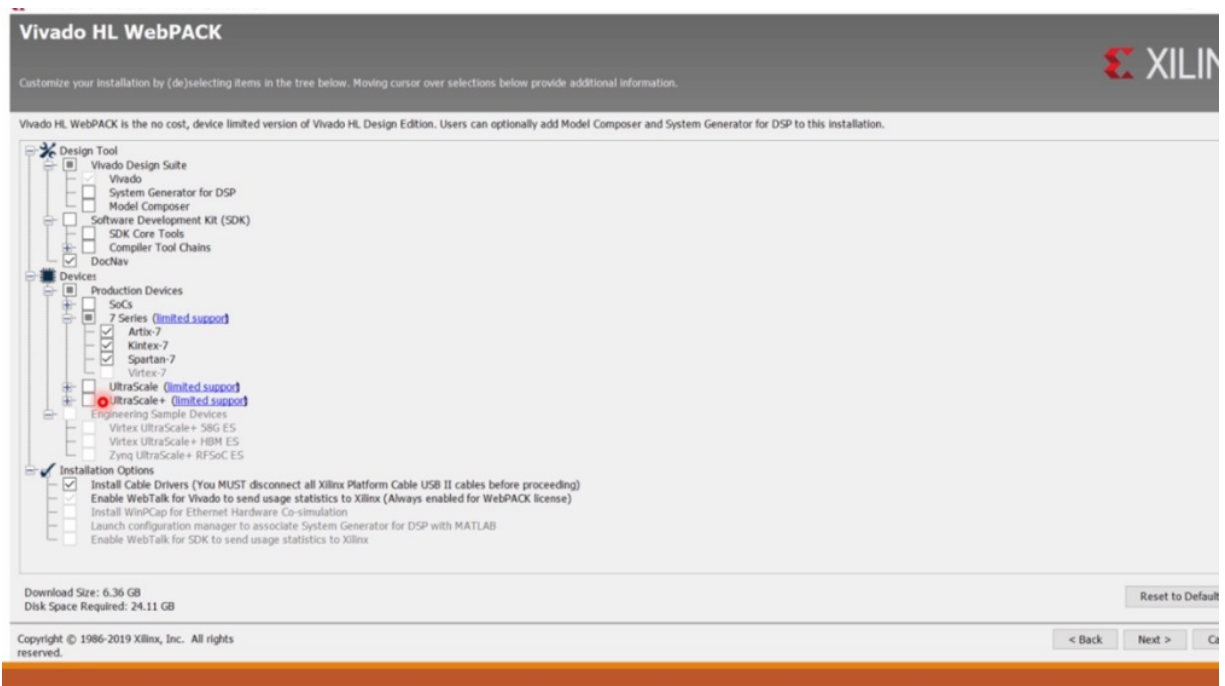
However the [REMOVED] is free so we can click on downloads and this is where you can get your installer.



As of this recording that's November 2019 the latest edition is twenty nineteen point two. And here we have the windows self extracting web installer which is not that heavy. It's sixty five megabytes but it will download all of the components you select. So this is the installer I recommend that you download. You can always download the whole thing which is 26 gigabytes. And it says here that is an all always installer but once again I recommend that you download the Unified Installer either for Windows or Linux and there's a step in this installer where you'll get to select which components you want.

SETUP_ THE VIVADO INSTALLATION WIZARD

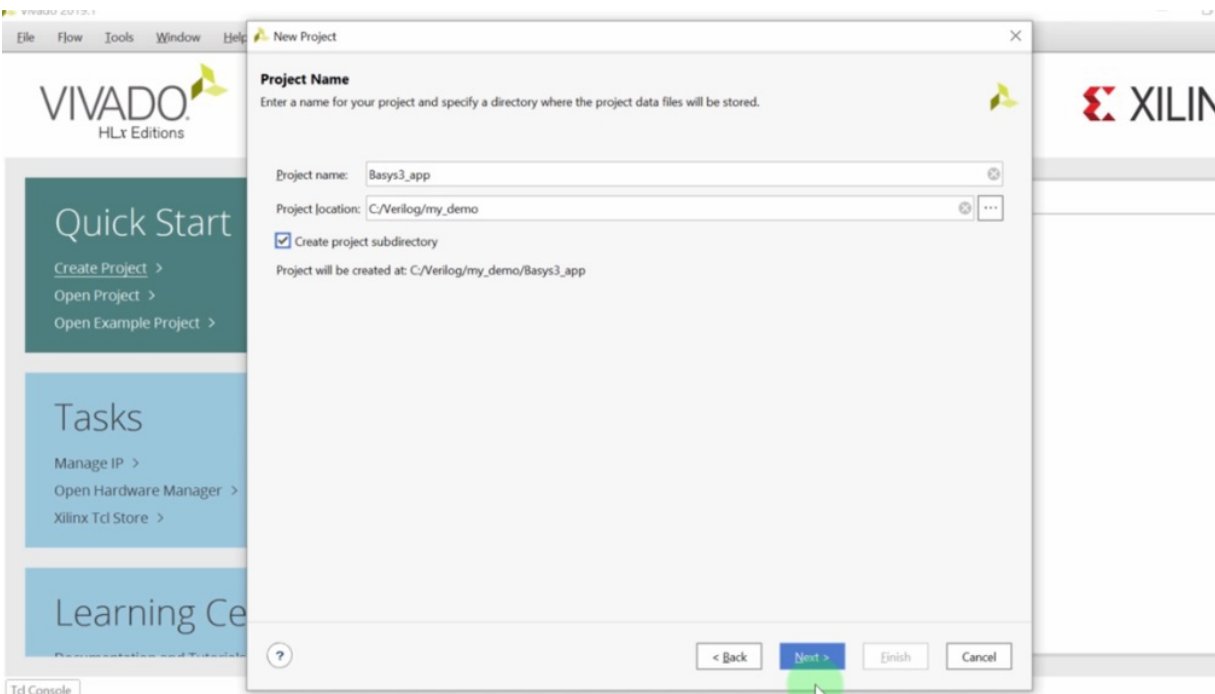
So this is the step in the vessel Design Suite installation wizard. We are interested in in here. You get to specify which elements you want to install. No notice several things in this window first. This is the installer for the vato HL web back. So Web Westpac is the free license you can use for development boards. Now the defaults for the elements in this list that you may install is not what's shown here. What I'm showing here is what I installed in my computer. And as you may know I did this to keep the download size and the required disk space down so the elements I've chosen here are the very very basic noticed that I'm only selecting for the designed to devalue which is not optional and I'm not selecting anything else not even the software development kit.



And that's because I am not intending to use any IP core for a soft processor. If I were I should selected then for the devices I'm only selecting the seven series which is the current the promoted series of FPGA. And some are available for the [REMOVED] and some are not for example the verdicts 7 is not available but the artists the kinetics and the Spartan are all available. I selected all three if you're working with a basis three board and you're only interested in it then you should only select the artists seven. Now I do have a Spartan seven board. That's why I selected this one and I intend to use a kinetics FBA in the near future. So I selected it too. And finally in the installation options notice that I selected to install the cable drivers because well you're going to need it when you download something into your FPGA. Finally I want to tell you that you're not supposed to install an S O C not for this discourse or the ultra scale FPGA s and these take up a lot of space. So that's why I didn't select them. So this is basically what I recommend for your installation but you're always free to use the default or to install more elements if you'd like.

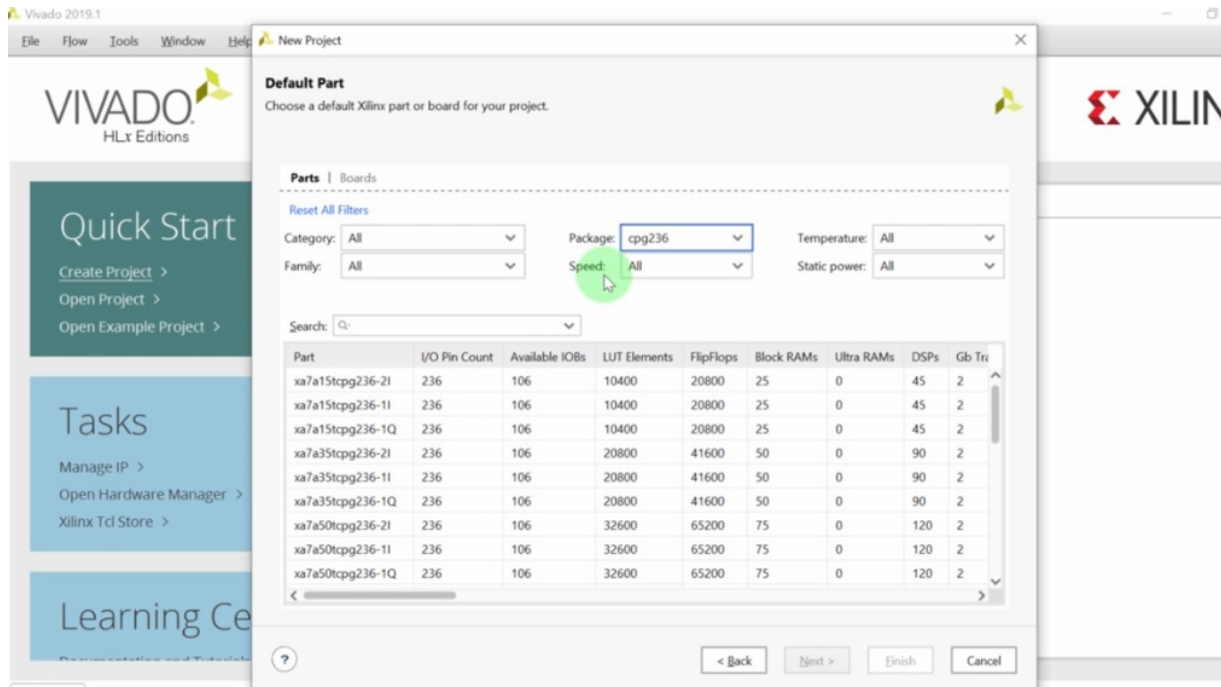
CREATING A PROJECT

So now let me show you a quick example of how to create a project on rebuttal. As you can see this is the welcome screen and you have some quick start shortcuts right here. So we'll select to create a project. This opens up a wizard and I will show you what to do on each step.

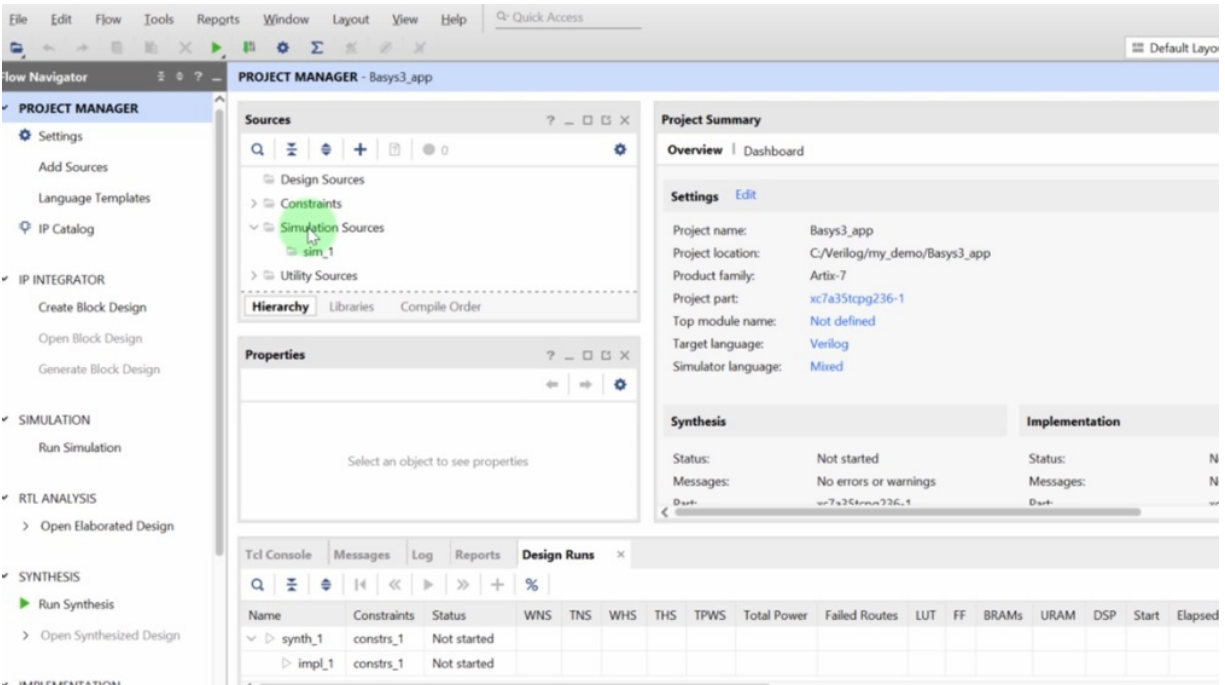


So you first have to enter the project name and the location of that project. And let me tell you that we've though has been historically quite picky on the directory naming. So you're not supposed to use special characters or spaces or long directory or file names. And so you are better off using shorter names and a simple directory structure. That's why I'm using C very low. And then another sub directory for example my demo. This is the directory where your project will be sent to. And I'll enter the name of the project. That's

name it basis. Three app and this is the directory that will be created. Very log my demo bases three app.



This is a very nice structure for v value. You also get to choose if you want to create a sub directory. So if I answer like this the project will go to my demo if I selected the project will go to its own directory named basis 3 app and inside it will have the whole project structure that's click on Next. And now we have to select what type of project we want. The objective of our project. So the default is an art deal project where you can get it to simulate and to implement. This is all we need for discourse. We have some other types of projects. And for the RTL project we can select whether or not to specify the sources. Right now at the wizard I will not create sources right now. So I will keep this checked and this part is kind of tricky. So you have to select the FPGA you are going to use but not just the family of FPGA. But the exact model and the exact chip you are going to use.

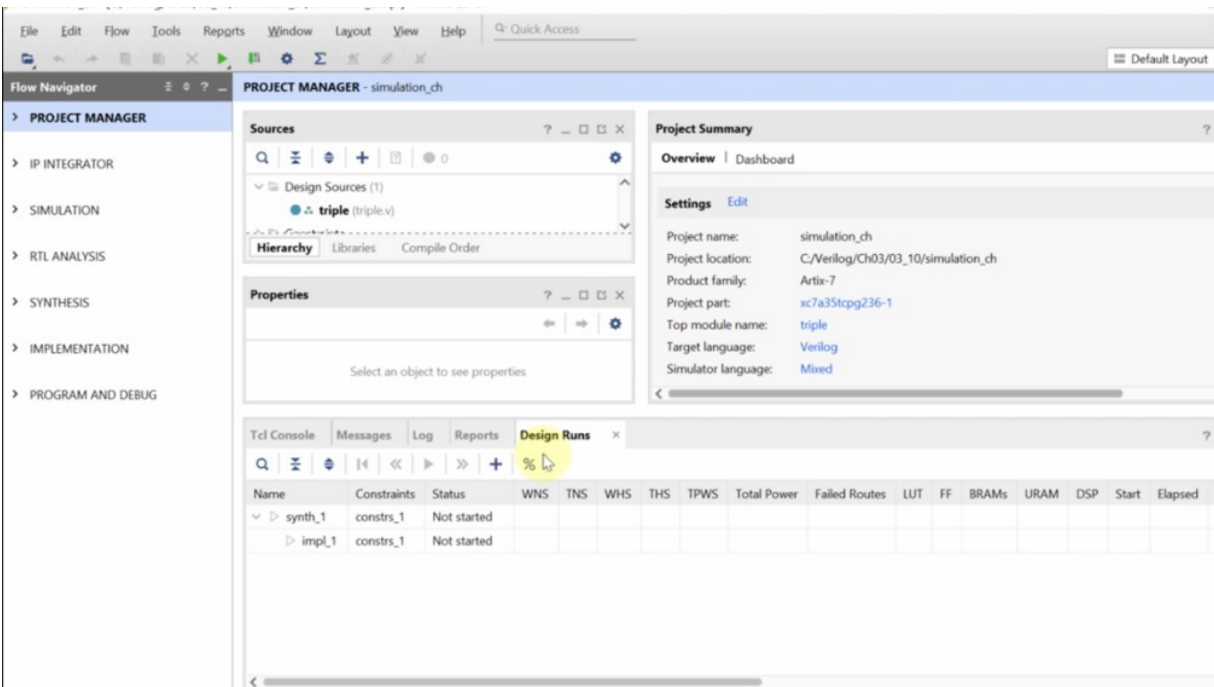


So this is quite lengthy as you can see this list is very long. That's why it has some filters right here to narrow down your search. Since we'll be using a basis three board you should know that this one includes an Arctic seven FPGA. So we could select from the category the automotive or general purpose FPGA is from the families we could select artists can context Spartan which are the ones we decided to install here and we have a lot of other options. So I have a nice shortcut for the basis three at least at this time. The Arctic seven FPGA on that board is the only one in the CPG 236 package. So if you select here you'll get a shorter list and the speed of that FPGA is minus one. So their son ranking of speeds in FPGA Ace and that's a number with some identifiers anyway. The one we'll use is minus one when we select that we get only three FPGA is the one we need is the one in the middle. This is the code you will see on the chip. It says X C 7 a 35 ti CPG 236 minus one. Yes I know this is terribly technical but that's what it is. So we'll select this one. Why. Because even though we may want to only simulate if we want to get it on the board we will need to know which FPGA this is going to. And this is needed for the implementation because the tool chain has to know what's available. So let's click next and that's it. We get a summary and we get to finish the creation of our new project once

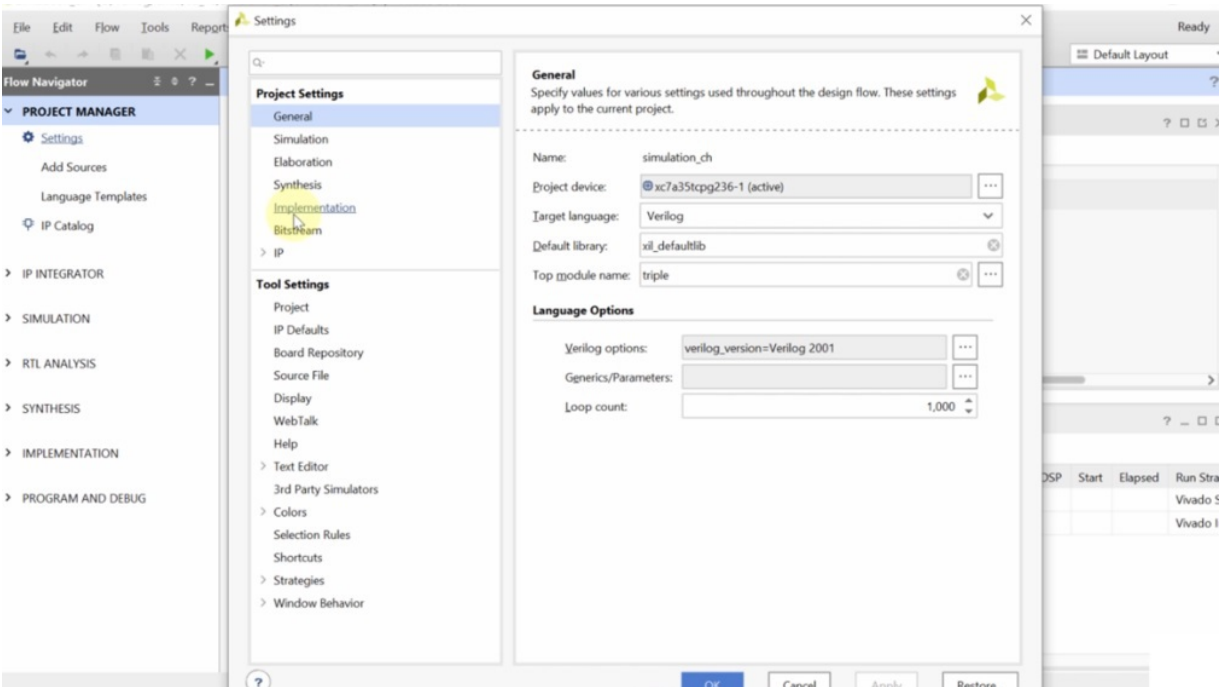
again. Since we have a new project here is the source structure. There are no designs. So if you want to add any files you can click on this plus button right here and you get to create design constraints which are timing constraints or pin assignments. You get to create your source files in very low or VHF deal and you get to create your test benches which are simulation sources. Let me cancel this and at the left we have the flow Navigator which has all of the steps for synthesis implementation generation of the bit stream file or for simulation. Anyway at this point you are ready to start working with your projects either for simulation or for implementation and download into a basic storyboard.

WALKTHROUGH_ PROJECT MANAGER

So that's almost it for the flow navigator. I didn't show you the first element and this first element is the project manager. So this is what's open right here.



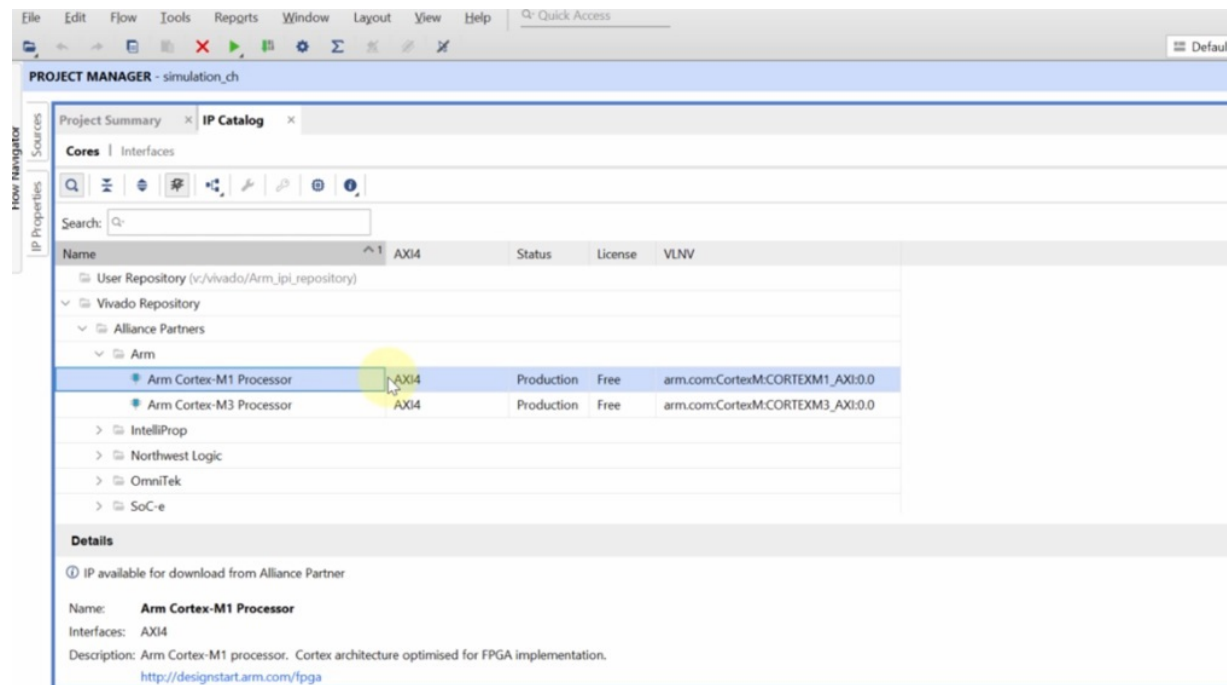
You will see this blue bar showing you which of all of these elements is open at the moment because these elements are views or perspectives in the idea. So let me open this part right here and we get to edit the settings for the project. So this project is for developing in very low for a specific FPGA and you get to choose all of the settings you would get on a regular i.e. next. We could add source files we could use templates.



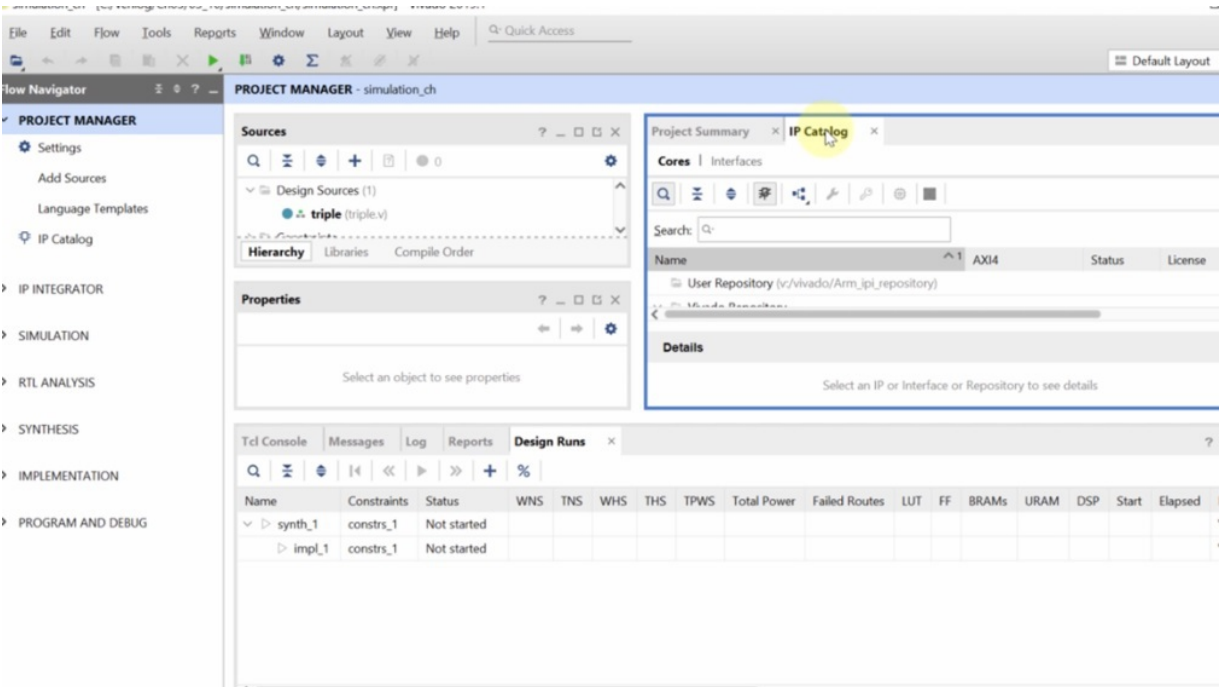
Here we have templates for very low. We have common constructs device macro instantiation for different FPGA. As you can see there is a lot of help for you to create your code.

WALKTHROUGH_IP CATALOG

Here we have the IP catalog. We can see all of the IP without using the IP integrator to maximize either of all of these windows. As you can see everything looks crowded here.



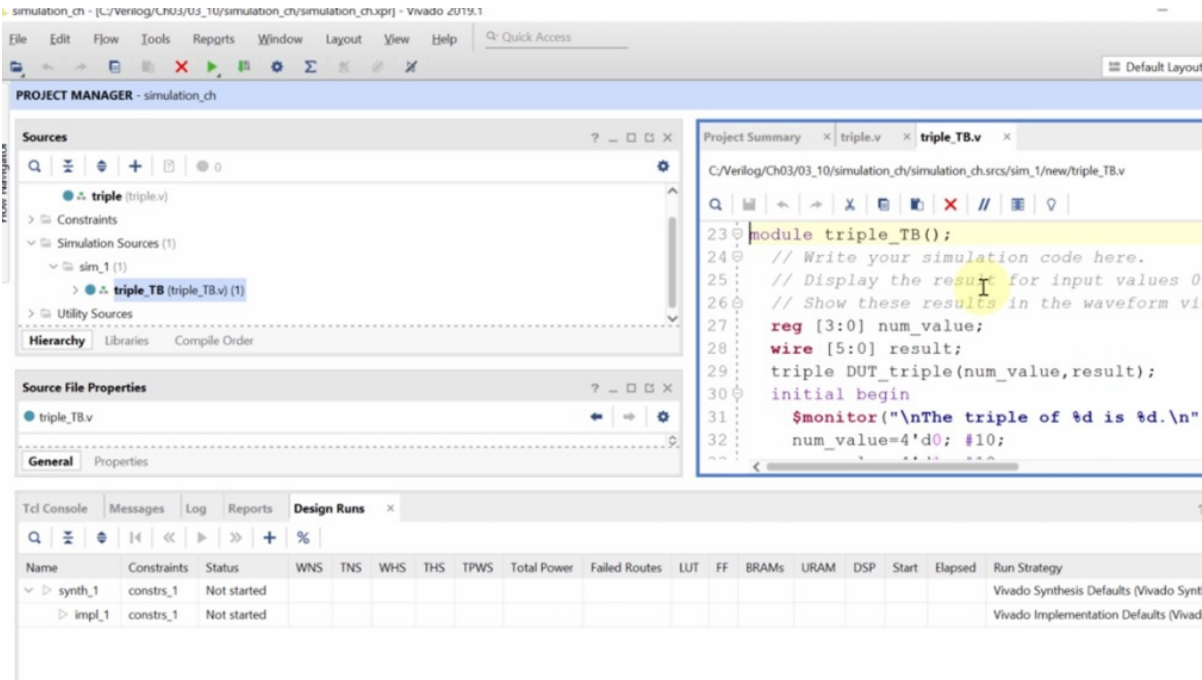
And this is something I don't like about Vevo but you can make peace with it by just double clicking on the tab and it will occupy this whole section. If the flow navigator bothers you you can always minimize it by pressing this button and it goes to a tab right here. So it's not that bad. And here we have all of the IP. You can see the Vive other repository. We have alliance partners like arm which provides the arm cortex and one processor.



This is only available as IP for FPGA. We have the cortex takes them three if you want you could use something from many other companies for example S O C E has some designs available you can use automotive and industrial applications a excite infrastructure which is an arm bus architecture. You have digital signal processing modules Embedded Processing you name it. So anyway this is where you get to choose all of the IP if you are willing to use it.

WALKTHROUGH_ SOURCE FILES

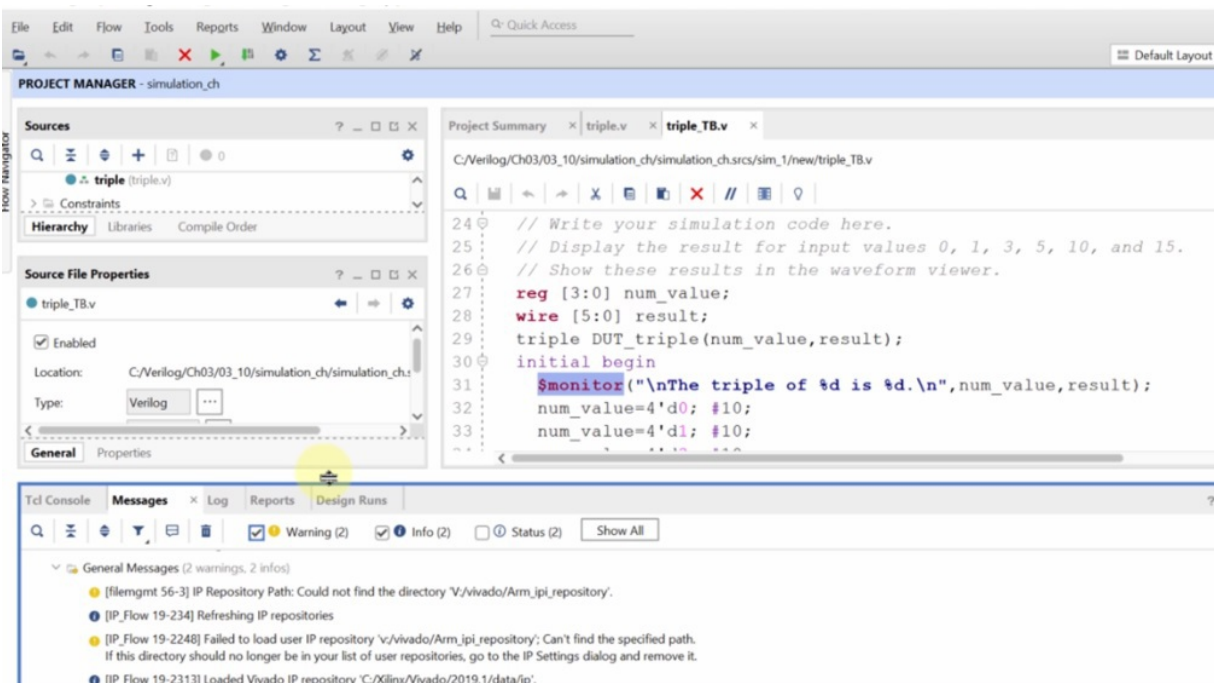
And now let me show you what do you get to use in the project manager. So here we have several views and several tabs. And let's start with the sources part. Right here we have a structure with these same sources. I have a source file named triple here which is an RTL design in very low that returns the triple of the input it has tripled right here input a the output is simply three times the input that's it. It's a very simple module.



don't have constraint files. This is only intended for simulation and here we have the simulation sources which has one file so here it is. It's called triple DP So here's the source file. It's called triple TB and it instantiate one triple module and then I am testing several values here 0 1 3 5 10 and 15. Notice that I am ending the simulation with the stop task. That is because site links uses a whole set of tools and it automatically sense all of the signals to its way form viewer. So you don't need to send it manually through a file. The other option you would have is to use finish and to send a VCAT file with all of that information to a third party way form viewer. But that's not the case with vato.

WALKTHROUGH_ CONSOLE OUTPUT AND MESSAGES

Next we have properties for each source file shown in this small section right here. As you just saw at the right we have the working part of the I.D. where you get to see the source code and for example the project summary right here with all of its information. And down below we get the usual thinks ideas provide. Here's the ticket console.

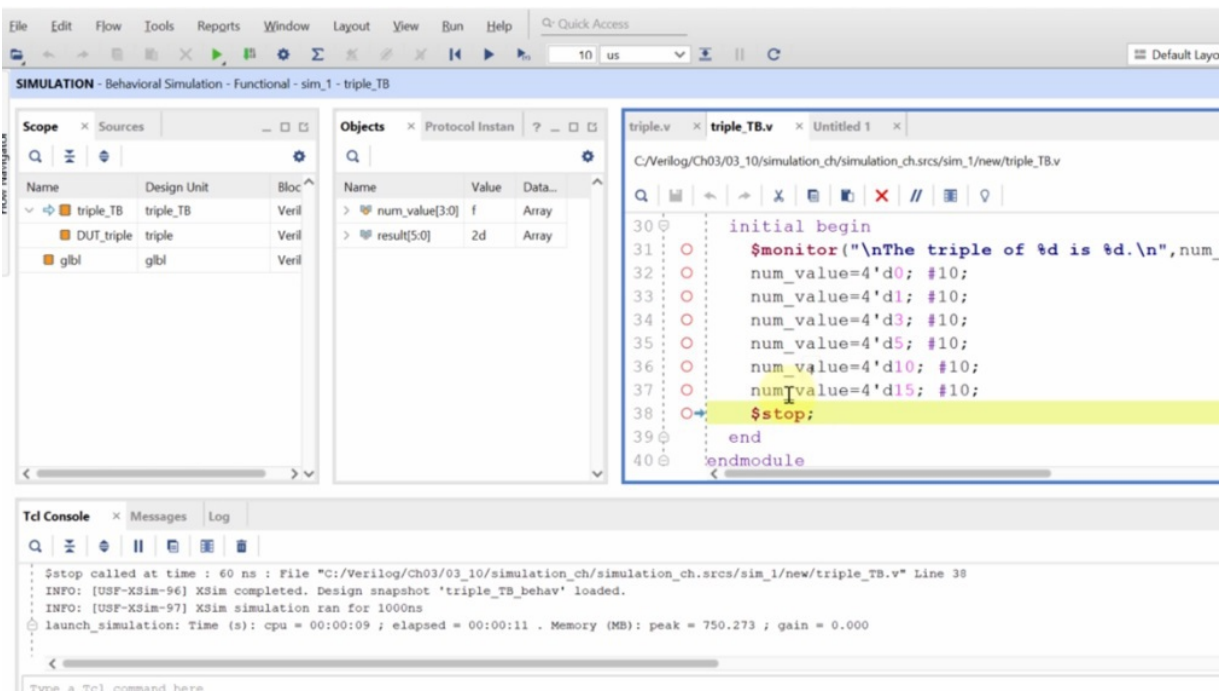


This console shows us the output for any simulation that you may send with for example the monitor or display tasks. We have a message as swindle with anything that requires our attention from

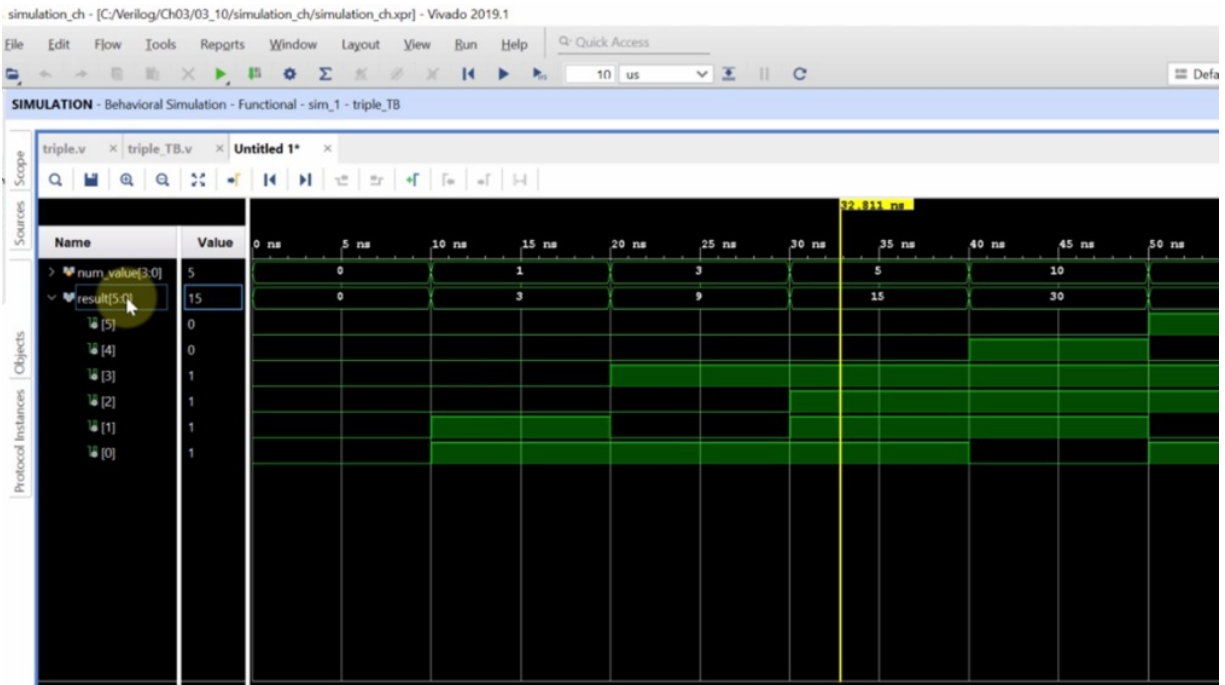
the tool chain. Here are some warnings we have the low window which shows us what's going on in synthesis implementation and simulation we get reports right here of anything that's been done and we have a report on the design runs that we have performed lately.

WALKTHROUGH_ SIMULATION

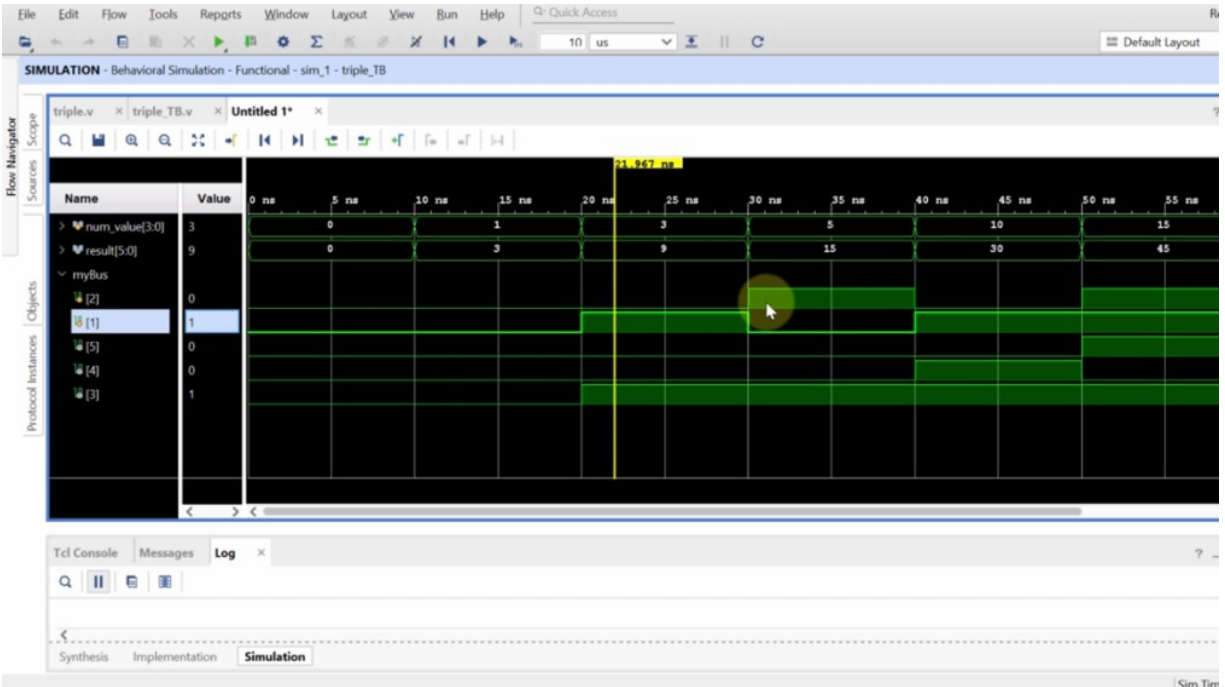
So we can run a simulation now. We just have to select simulation and run simulation. It's that easy run behavioral simulation and now the perspective will change.



We will not see the project manager but the simulation view let me hide the flow navigator. And here is the simulation that has already stopped. But the tool hasn't stopped. The simulator hasn't stopped. Just the simulation this untitled tab is the way form viewer and here we have all of the simulation. So the zoom level here is a bit tricky. What I always prefer to do here is to press this button zoom to fit and here it is. We have zero three times this zero one three times is three three times three is nine.



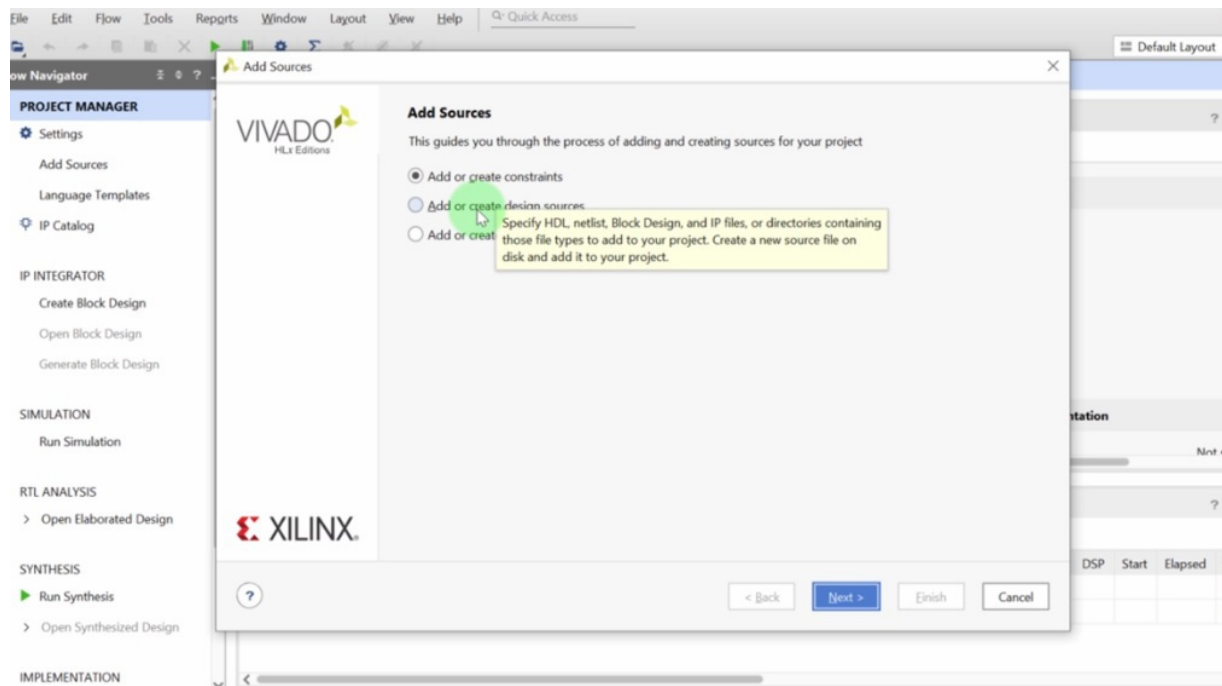
Three times 15 is 45. And if you find it a bit annoying to see Hex here you can always select your data right click and change radix to unsigned decimal and here we have it three times five is fifteen three times ten is thirty and three times fifteen is forty five. You can always expand buses like this signal result which is six. It's long and here are the individual bits with the whole bus interpreted in decimal we can do the same for the four bit input which is this one right here and you get to use these cursors This is the default cursor and the value of each signal at the cursor is shown in this column right here named value.



You may also create virtual buses right here by right clicking and selecting a new virtual bus let's call it my bus and we can add signals to it you can do that by selecting objects right here and let's say I want these two bits added to my bus and I don't know this that's also added to it you can move your signals up and down and here's what you would get for that bus you can collapse at Oh I'm sorry I didn't include these inside the bus you just drag them and drop them now they are part of this bus and since these signals are part of the bus the whole bus is being interpreted right here as you may notice it's shown in hexadecimal again if you want decimal right click radix answer in decimal and it's showing whatever signaled you may want to produce here this one has no meaning to me but there you have it.

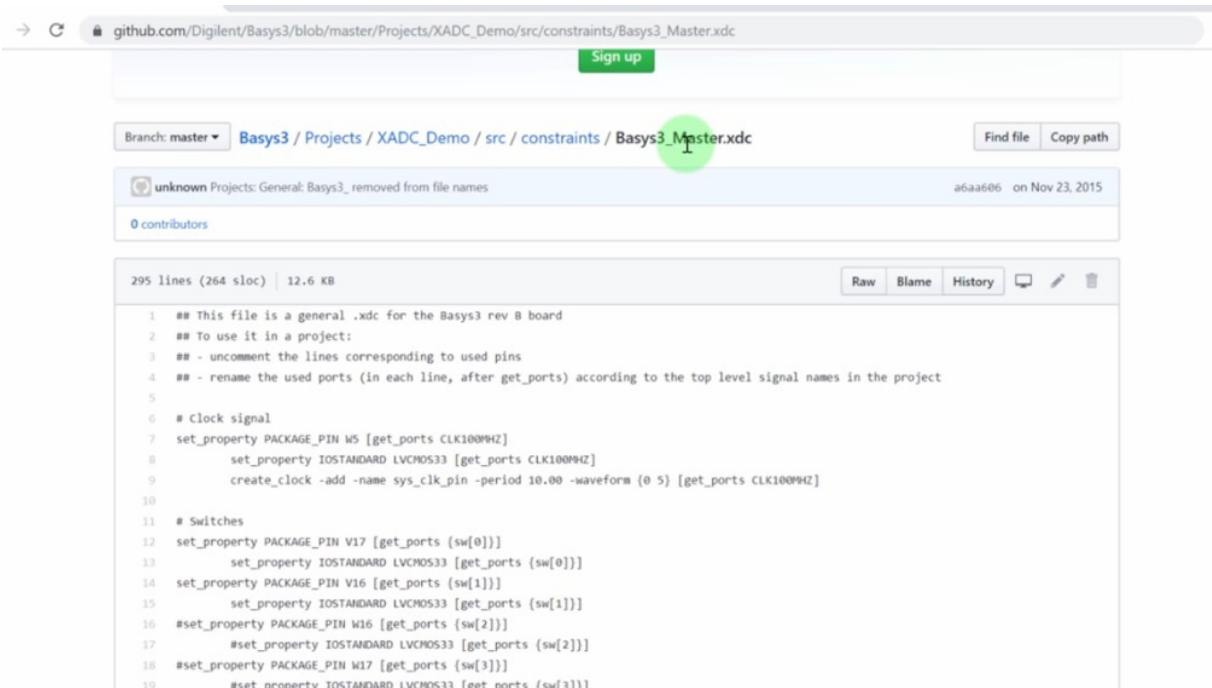
WALKTHROUGH_PIN ASSIGNMENT

Now, to get this design into a basis three board, we need to run some steps first. We need to add constraints file, which will make the PIN associations between the input output ports in our top design and the actual pins in the FPGA integrated circuit. So to add that file, we can select this plus sign right here to add sources and we get to select if we want to add or create constraints right here, we can also add design sources or simulation sources.



Basically, these are the PIN associations. These are the modules we are going to use and these are the test bench modules we will use in simulations. So now let me just create a file. It will be called basis three, that SDC, SDC is the extension for sailing's design

constraints. And let me click on finish. So if we look at that file, it will be empty, but luckily vigilant has already provided a next D.C. file for your designs so you can look it up online. You can make a Google search like this basis three dot SDC currently that is on GitHub. So you can select this result.



The screenshot shows a GitHub repository page for the file 'Basys3_Master.xdc'. The page includes a 'Sign up' button, a breadcrumb trail 'Basys3 / Projects / XADC_Demo / src / constraints / Basys3_Master.xdc', and a commit message 'unknown Projects: General: Basys3_ removed from file names' with commit hash 'a6aa606' and date 'on Nov 23, 2015'. The file content is displayed as follows:

```
295 lines (264 sloc) | 12.6 KB
1  ## This file is a general .xdc for the Basys3 rev B board
2  ## To use it in a project:
3  ## - uncomment the lines corresponding to used pins
4  ## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project
5
6  # Clock signal
7  set_property PACKAGE_PIN W5 [get_ports CLK100MHZ]
8      set_property IOSTANDARD LVCMOS33 [get_ports CLK100MHZ]
9      create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports CLK100MHZ]
10
11 # Switches
12 set_property PACKAGE_PIN V17 [get_ports {sw{0}}]
13     set_property IOSTANDARD LVCMOS33 [get_ports {sw{0}}]
14 set_property PACKAGE_PIN V16 [get_ports {sw{1}}]
15     set_property IOSTANDARD LVCMOS33 [get_ports {sw{1}}]
16 #set_property PACKAGE_PIN W16 [get_ports {sw{2}}]
17 #set_property IOSTANDARD LVCMOS33 [get_ports {sw{2}}]
18 #set_property PACKAGE_PIN W17 [get_ports {sw{3}}]
19 #set_property IOSTANDARD LVCMOS33 [get_ports {sw{3}}]
```

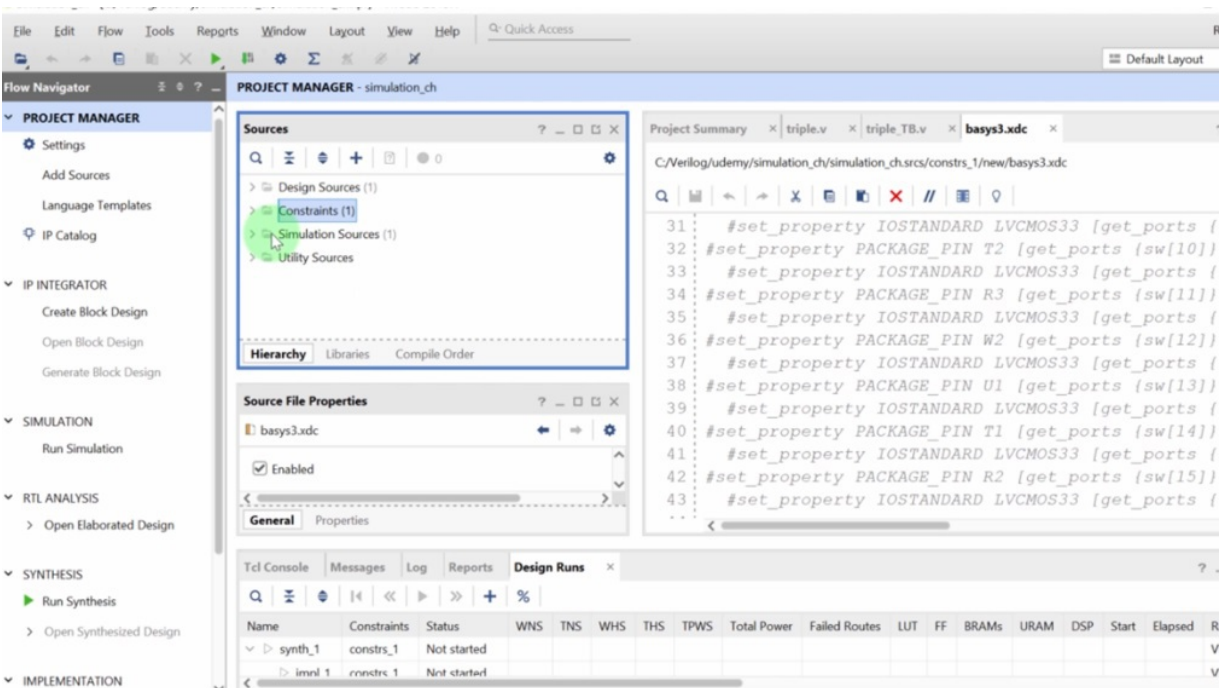
And this is the file basis three master SDC. So what I'll do is copy the whole contents of this file and show you how to modify it to make your application work with the bases three board. So I will copy right here. And they will be based here. Now, this file is originally intended for you to uncommented the lines that are related to your design. So, for example, right here, the author has uncommented these three lines which are enabling the clock signal. But I will come in time out. You know, the original master file comes with all of the lines coming out. So I apologize for this, but I will use the four rightmost switches to enter the Knebel for our triple module.

```
5
6 # Clock signal
7 # set_property PACKAGE_PIN W5 [get_ports CLK100MHZ]
8 # set_property IOSTANDARD LVCMOS33 [get_ports CLK100MHZ]
9 # create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports
10
11 # Switches
12 set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
13 set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
14 set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
15 set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
16 #set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
17 #set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
18 set_property PACKAGE_PIN W17 [get_ports {sw[3]}]
19 #set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]
20 #set_property PACKAGE_PIN W15 [get_ports {sw[4]}]
21 #set_property IOSTANDARD LVCMOS33 [get_ports {sw[4]}]
22 #set_property PACKAGE_PIN V15 [get_ports {sw[5]}]
23 #set_property IOSTANDARD LVCMOS33 [get_ports {sw[5]}]
```

And so I will use these two lines that are related to switch zero. As you can see, the syntax in this Tzedek file called the Get Bortz function, which fetches for this symbol in our source files. So whatever is called S.W. zero in our design will be mapped to bin V 17 in our FPGA. We also get to specify the standard, the input output standard for that pin. And so this is low voltage seamless at three point three volts. I will enable then all the switches from S.W. three down to S.W. zero. And next, I will enable all leads from LED Zero all the way through Leidy's four, because, as you remember, our triple module has actually it has six bits, so. I will enable them until our LED five, the rest will be common throughout. And that's it for the Tzedek file. Once again, I will only use switches zero through three and early this zero through five, the rest of the file should have all lines commented out, as I've just made sure here.

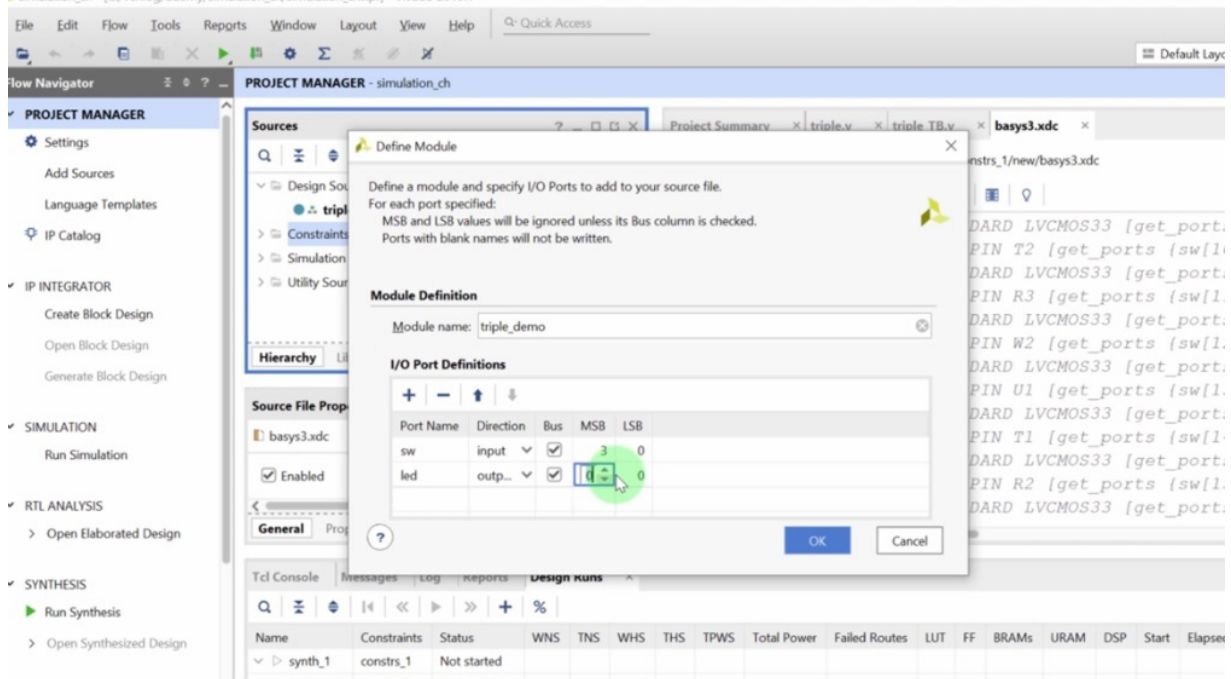
WALKTHROUGH_TOP MODULE CODE

So at this point we have our design sources which is this triple that the file with only the triple module. We also have our simulation sources which is triple TB a test bench for that triple module and we have our constraints file which is the basis three dot X DC file remember this one makes the pin assignments.

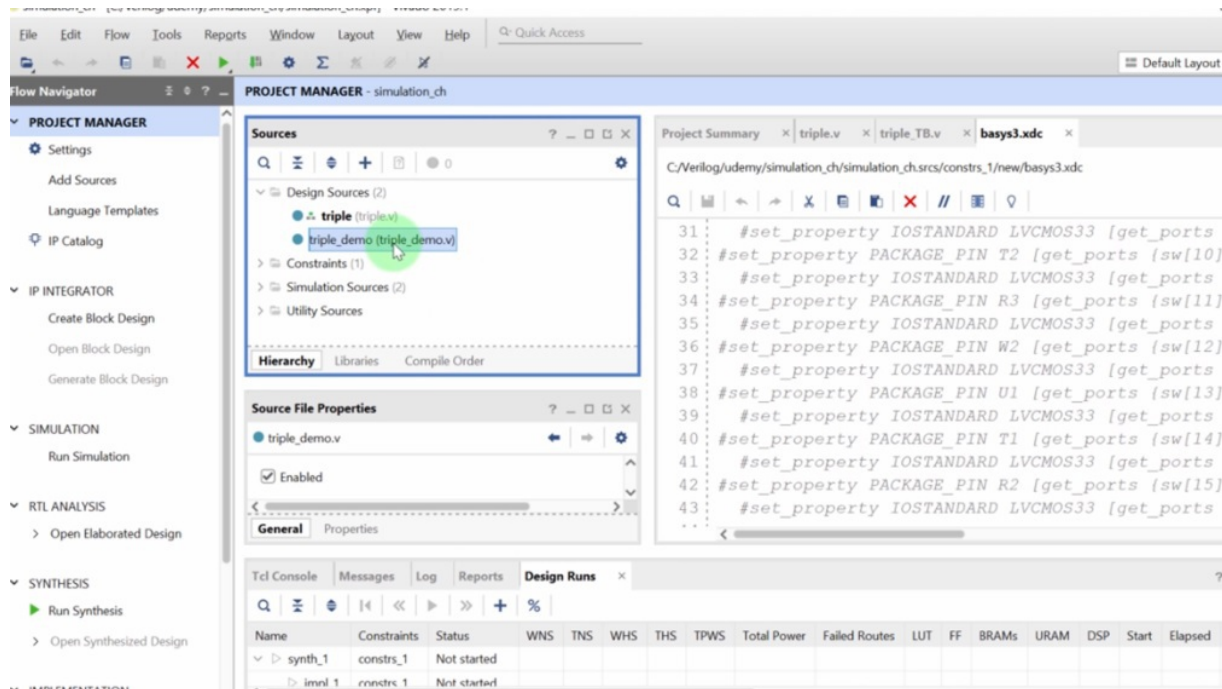


We decided to use. Now we need to write our top module which will be the only module downloaded into the FPGA. This module has to instantiate whatever we need in our application. So let's start by adding sources. I want you to notice that this source will go into the design sources section right here. So let me add a new source and we have to create a design source I'll click on Create file and I will

name this triple demo. Now when you create a dot v file you will get this dialog box which asks you to enter the IO port definitions. This is to make a nice skeleton code for you but you can always type in your code manually.



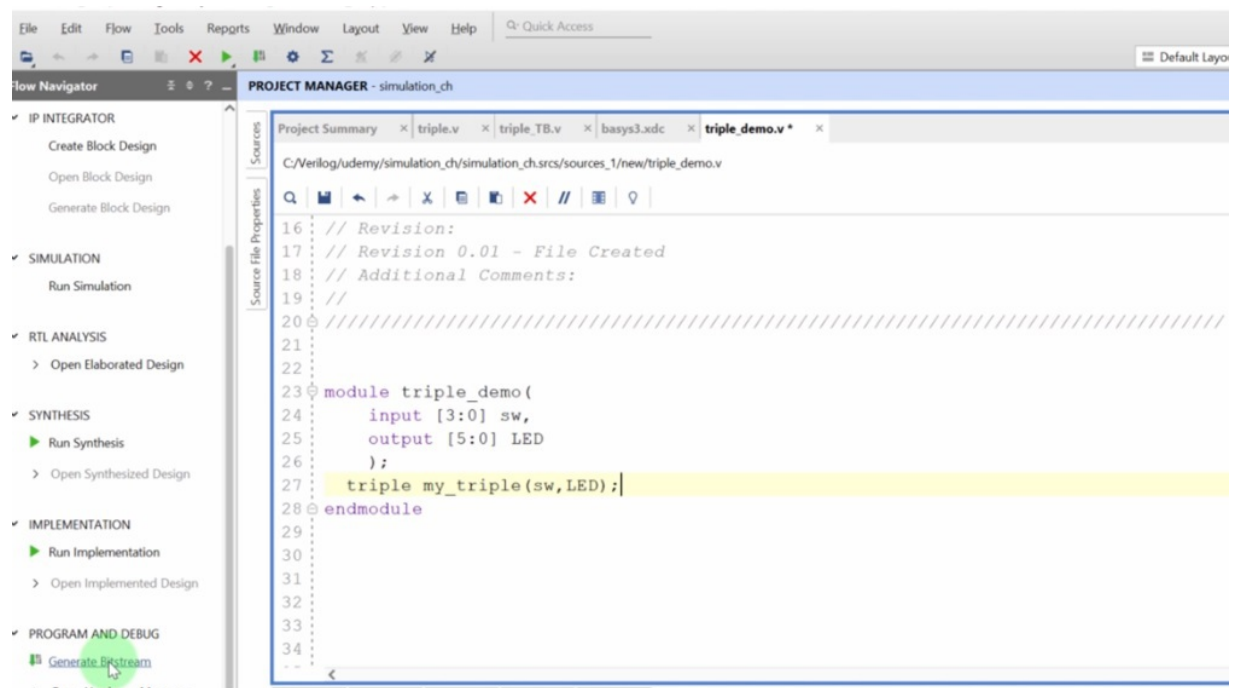
So we'll have four bits for the input and six bits for the output remembered that are input will come from the switches and the SDC file is using the name S W. So I will name this as W and it will be an input bus. This is the most significant bit 3 and 0 the least significant bit. And then the output will be sent to the LCD. I believe that was called a lady. It's an output port and it's a bus from 5 through 0. So when I click okay that file will be created and here we have it tripled demo. You may want to notify v rather that this is your top design and not this one. So that's what this little icon right here means that this is the top module. However developer keeps an ongoing screening of the files so that it recognizes which module instantiate which module.



So it figures out which is the top module. Anyway we could right click on this one and select set a stop. I won't do it because I want to show you how the battle is smart enough to figure it out. Anyway I will open it and here it is. This is the skeleton created by V vital for you. So let me just maximise this and this will be very very easy. Okay. So I will just instantiate a triple module. It will be called my triple the input will be my switches and the output will be my lcd. Let's just double check that the name of the early this array is correct. This is in the SDC file and they are indeed called LCD. But uppercase LCD. So let me correct that right here. This is called LCD so you know that very low is case sensitive. And there you go. This is my top module as you can see it's a wrapper for my triple module. It ultimately depends on how you are designing your system. But typically the top module just instantiate a couple of modules and makes some connections and it doesn't have much circuitry or logic in it.

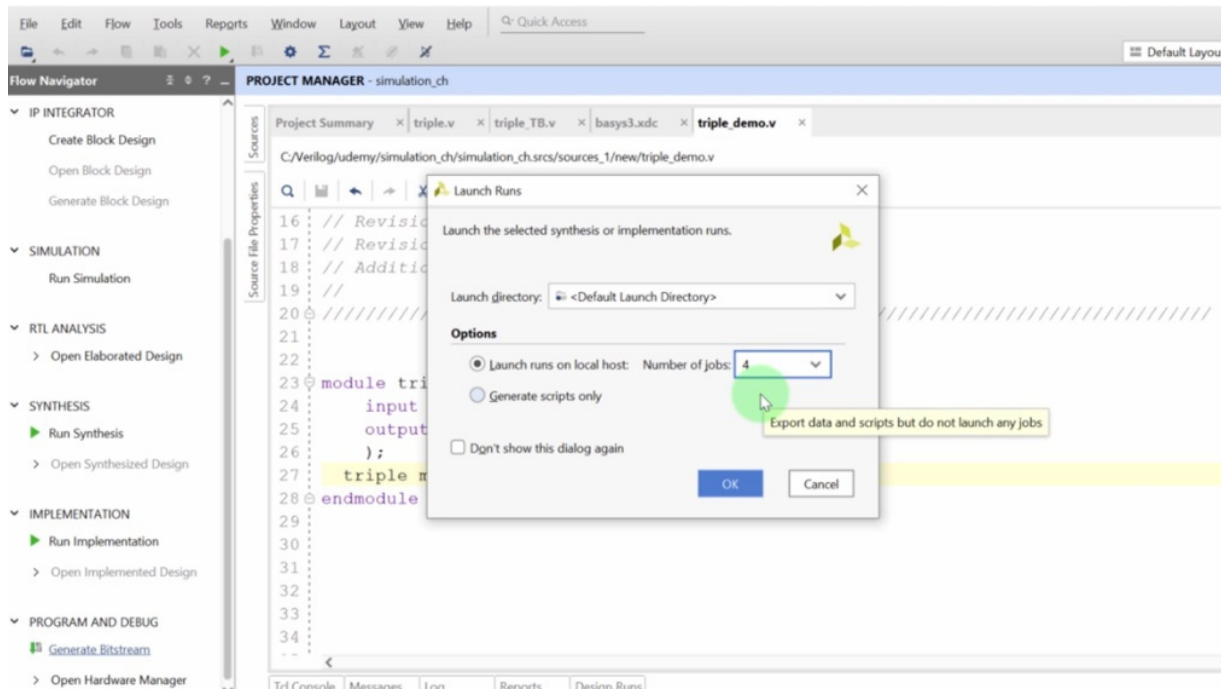
WALKTHROUGH_ BITSTREAM FILE GENERATION

So that's it. We are ready to synthesise and implement this design to finally generate the programming file which is called a bit stream file inside links jargon and just like with Cordis at the left we have all of the tasks you can perform to get your project to produce its purpose.



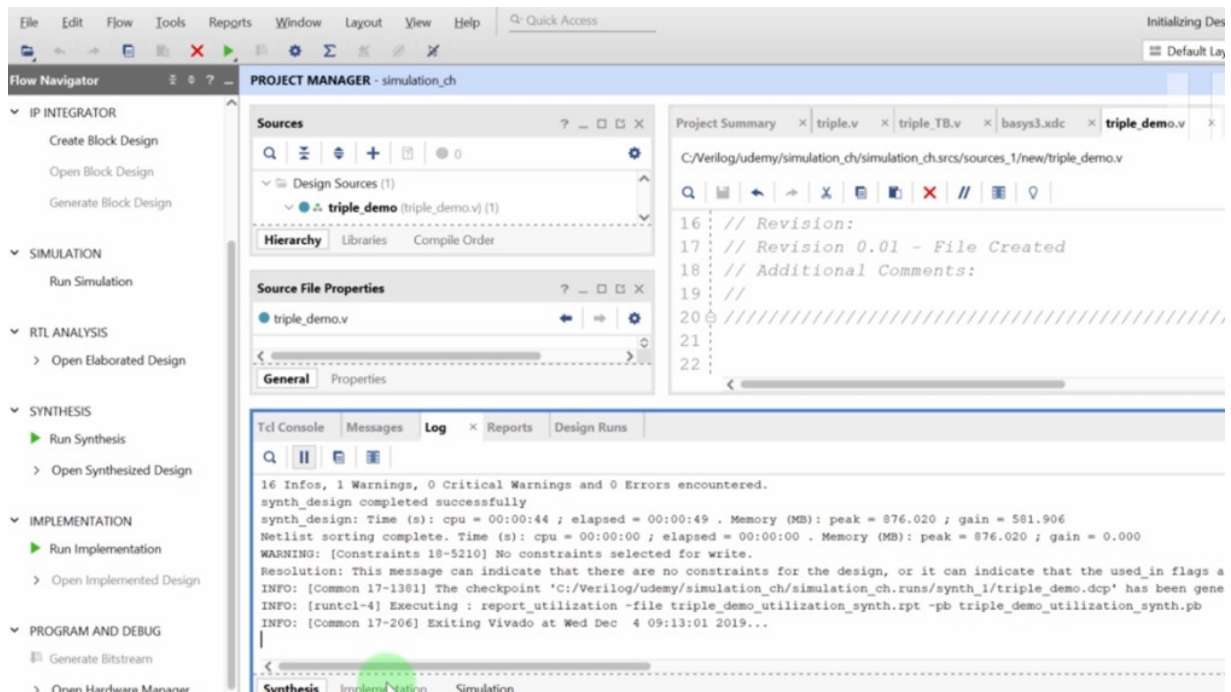
So you may achieve a simulation or you may even implement and generate the programming file. All of that is selected in this part at the left. And again similar to quarters you can simply run the last task you want to run your objective and it will run. All that's needed in order to complete that task. So we need to generate a bit stream file.

That's the ultimate thing we want. And lastly we want to open the hardware manager where we will download the application or rather the design into our FPGA.

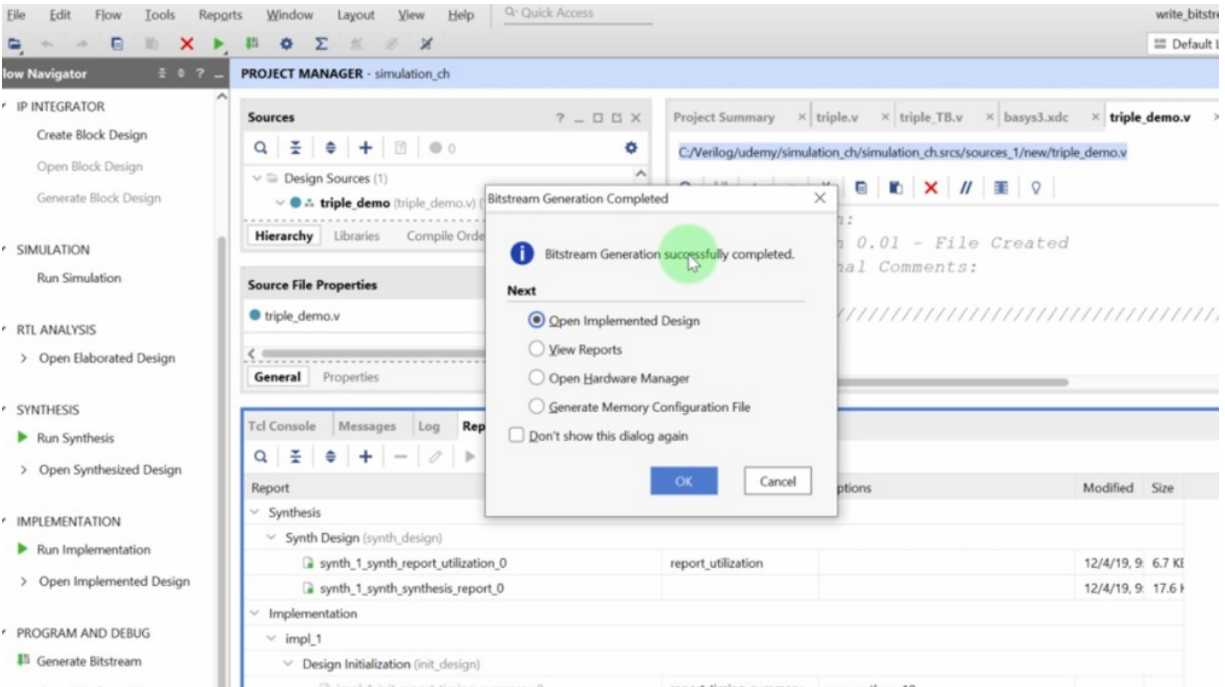


But in terms of the tool chain the last part is to generate the bit stream. So I will click here. So I have to save my files. And here we get a warning that says that no implementation is available because this is the first time we are going to implement this design. And so this dialogue says that what we intended to run which is generate bit stream will automatically start when synthesis and implementation is done. And that's what you would expect of an idea to so well click on yes and all of this process is quite lengthy because it's on a path to a compiler tool chain. And actually it's a bit more complex than a compiler tool chain because this process does not translate to assembly language instructions but to routing off hardware in a chip. So this is quite consuming in computational power. That's why this dialogue is asking us to select how many jobs we want to run and by Jobs This means the processes that are going to be assigned to CPE use since modern computers have multiple use then you are given the choice to select how many of these you want to assign to

your use. It's often recommended that you leave the default right here for me it's four and this number comes from the installation when your operating system notified vital that it has four processors available anyway let's click OK and wait for a while you can always see what's going on in this section at the bottom. Here we have the design runs which is showing us what's going on in a nice stable. We have reports here where we will get what resulted from every step. We have the log tab right here which is telling us all of the output that's sent from the tool chain we get the messages arranged in a nice tree right here so that we can read what needed our attention warnings and errors. And we have the tickle console which serves as an output window.



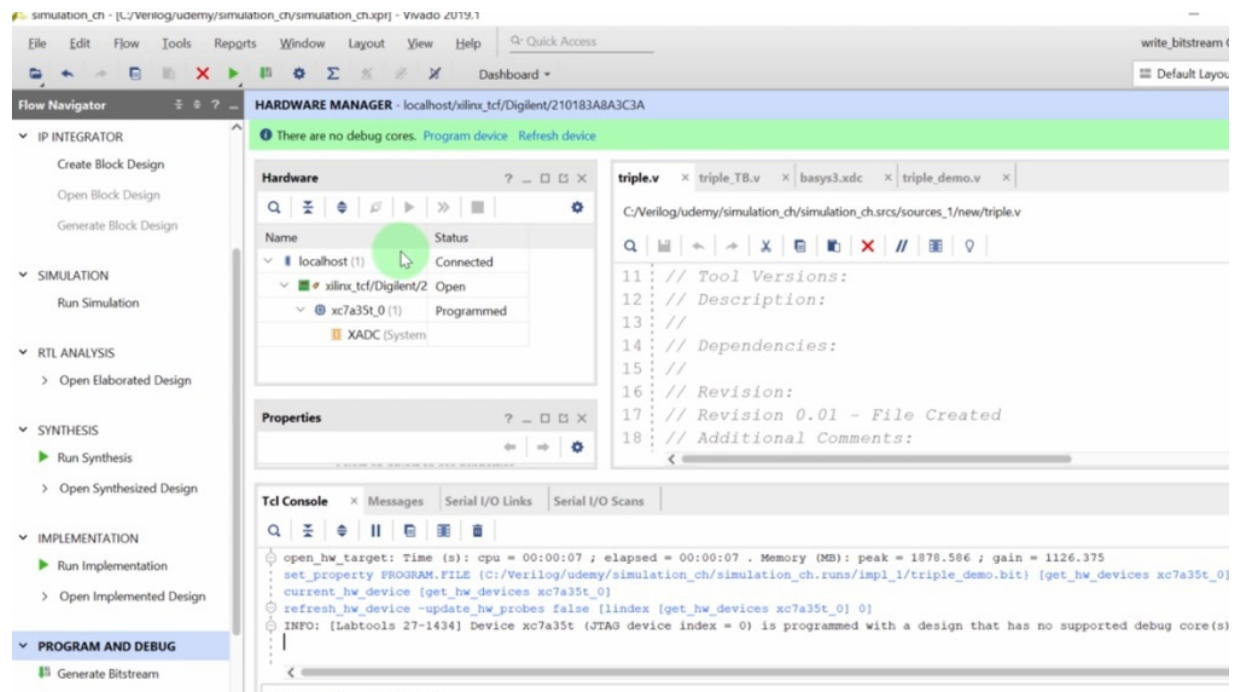
This is where your output from tasks like display and monitor goes to and by the way the log tab has several taps at the bottom. It has one for synthesis one for implementation and one for simulation. So it's all arranged in a tidy way. Let's go back to the design runs. And as you can see synthesis is already complete it has occupied three lookup tables zero flip flops and the implementation is currently running and you can also see what's running here at the top right.



It says running root design and you can always cancel whatever is running and now we wait and apparently all went well. It's this bit stream generation successfully completed so after each step of the way you get one of these dialogues that asks you what you want to do next. You always get to disable this question but I like to keep it because I like to do different things for different projects. So we get to open the implemented design to inspect what's going on. We can also view the reports we can open the hardware manager which is what we intend to do right now or we can generate a memory configuration file.

WALKTHROUGH_ THE HARDWARE MANAGER

Notice that the blue bar on top says HARDWARE MANAGER. So every part right here at the left is a different perspective at which you can look at your project.



So the hardware manager is concerned with the connection between the bases three board and vato. So it says on connected and this green bar says no hardware. Target is open. We have to connect the board through a USB port. Let me do that and we have to connect the board logically to vato. So we get a shortcut right here that says Open Target and we get to choose what to do. The best option is auto connect. It does everything for you and after some handshake you'll see that the hardware manager has this section right here

named hardware showing which board is connected to your computer. Let me maximize this so it says that we have a sailings vigilant board. This is the name of the FPGA chip we are using and we are ready to make a connection. Again we get a nice shortcut in the green bar. It says there are no debug project because we are not using a debug core. We are not debugging from the hardware but we are ready to program the device. This is the shortcut we want. So let me click on it and we get to select which bit file we want to send this bit file is triple demo that bit. This is organized in my project directory structure. And if we wanted to perform some debugging with a debug core we would enter that file here. But anyway our output is in the bit stream file field right here and we are good to go. So let me click on program and now my FPGA is programmed. So let me show you what's going on on the board. As you can see all of my switches are down and if I enter for example one you can see that we have a number three on the LCD. This is binary. And let me enter for example two we'll get six if I enter seven we get twenty one that's sixteen plus four plus one.



That's 21. Remember that we have four bits in our input so I can enter any for bid number for example that me and 13 and that number we see on the LCD is 15 shifted one position to the left. So that's two times 15 or 30 which is the result we were expecting the triple often is 30. Let's see the highest value we can enter. That's 15 times three is supposed to be forty five. So let's take a look at the bits we have a 1 in the most significant bit which is 32 plus the lower four bits are 12 plus 1. That's 13. So 32 plus 13 equals forty. Just as we expected Now remember that our attention is not specifically on the hardware. After all the displays are showing all bits kind of dim and we could have avoided that. But we are more interested in the tools that are available in the vato software. And I hope you got a nice grasp of all of the process and of the many tools you can use. These are only the basic tools to get a simple design into a board and there's a lot more for you to play around with.

LABSLAND

Now it's time to talk about one of the coolest tools I've seen online. I'm talking about lapse land. Let me tell you what we'll see in this project. First I'll tell you all about this too and then I'll tell you about the steps of the process that apply to lapsed land. So I'll tell you about the setup you need to take in order to use this wonderful tool.

In this lesson: LabsLand

- About LabsLand
- Setup
- A Quick Walkthrough
- Code Example
- Implementation

I'll give you a quick walk through I'll show you a code example and an implementation example. Now since this is a remote lab tool it only supports implementation. There is no simulation available because you're not simulating you are running into the actual hardware in real time. You are watching the hardware running in front of you at a remote site. So let's get started.

ABOUT LABSLAND

So first let me tell you about lapsed land this is a remote lab online tool. And this is part of a big thing that's been going on online lately I may have mentioned that I am an educator and I teach at the University undergrad and graduate levels and we have been developing online programs. And one of the biggest challenges we've faced is getting our remote students to have the laboratory experience they would have if they would attend to our facilities. And so we have used remote desktop tools with a camera on it so that they can see the hardware and getting a two way communication is a bit of a challenge. And also scaling those labs is kind of a problem. So we found labs land which is a company that provides this online tool for you to get access to hardware remotely. You can handle schedules and of course multiple users. So this tool enables you to use a wide variety of expensive hardware at a low cost. Yes this is a commercial tool but you may want to take a look at their Web site which is let's land dot com and you'll see that it's really really cheap and you may even get a nice discount as I'll tell you a bit later so this tool offers real laps from schools and universities all around the world. And the way it works is that some university can subscribe their laboratories and make them available for labs then to offer to the world.

About LabsLand LabsLand

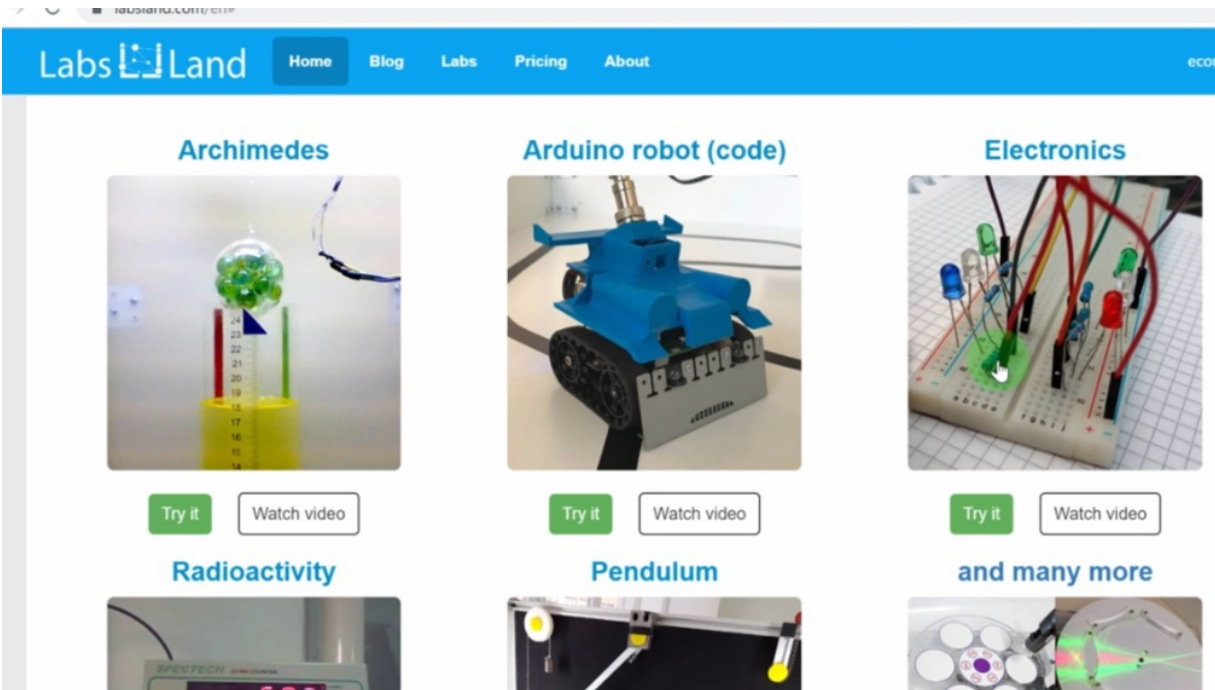
- A Remote Laboratory online tool.
- Enables you to use a wide variety of expensive hardware at a low cost.
- Offers real labs from schools and universities around the world.
- Not limited to FPGAs.
- Basic Electronics, Arduino, Physics, Biology, Optics, Radioactivity, etc.



Here we have a nice map taken from left lance web site which shows the locations of the labs that are currently attached to their system. And so when you choose to use some board you will get assigned one that is located in one of those laboratories. Again this is an exciting development of the Internet. And one more thing is that this is not limited to FPGA S. as you'll see a bit later It supports a lot of laboratories. It has basic electronics. It has a couple of our Dino labs. It has a lot of physics labs from Kenny Maddox to biology optics radioactivity. There are tons of labs available so although this is a commercial application and you are not required to buy anything to be successful in this project I do recommend it. It's a very nice alternative to acquiring a very expensive FPGA board. And remember you'll get to work with many boards. So it's actually an alternative to buying all of the boards you intend to use. It's really nice. You should give it a try.

WALKTHROUGH

Now let me give you a quick walk through a flaps land. So this is the home screen you'll get once you've logged in. Now let me scroll down to show you some of the elements you'll find here. First we have three links one for the remote labs one for a demo if you want to try it out and you can also attach your educational center your own labs into lapsed land so that you can offer your laps remotely to the world and we are interested in using the remote labs for discourse. So let me scroll down a bit more to show you what you can do here. Here's an explanation of how the remote labs philosophy works.



And here we have some demos as you can see we have an Arduino robot here which is working at a site and you can download your programs into it. Here we have an electronics lab. This is very nice because it consists of something electrically similar to an FBA but it's

not. It has an array of elements a big array of elements with relays so that you can design your own circuits and test them because it has all of the instruments and the circuitry you'll need to create pretty much any circuit you can come up with once more. This is for electronics there's a radio activity lab available there's a pendulum one this is for physics of project and there are lots lots more. And just think about it for a minute the fact that you have labs for say radio activity available to anyone makes this to a very powerful and a very clever use of the Internet was more scrolling down you'll see that you can even be a consumer of lapsed land as an institution. So you can enroll your students and you can set up scales for them to work on the labs. So again this is a very clever and very well designed system and it also supports several learning management systems like modal or sky canvas and many more so you can most likely attach your own system into it very very easily here's a promotional video and there you have it for the home screen. So now let's take a look at some random lab. Let me show you the. They are doing a robot lab for example I just clicked on try it and you can click on View experiments. Here it is. We have the Arduino I.D. right here and we have a robot example. Let's take a look at it so in the interface you get some limited time to use the robot.



Leave now

Arduino robot



Your own programs

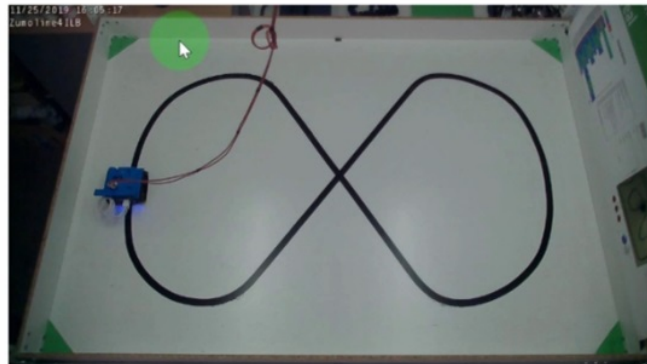
In this example, you cannot send your own code. This can be done in the Arduino IDE laboratory (to program through standard code), or to the Visual Blocks laboratory (to program through a visual language). From those, you will be able to send the code you write to the robots. In this session, you can still send the example programs below.

Example programs

Line follower

Program into robot

In the beginning, you can move the robot wherever you want. Then, you can use the left panel to upload a program to the robot.

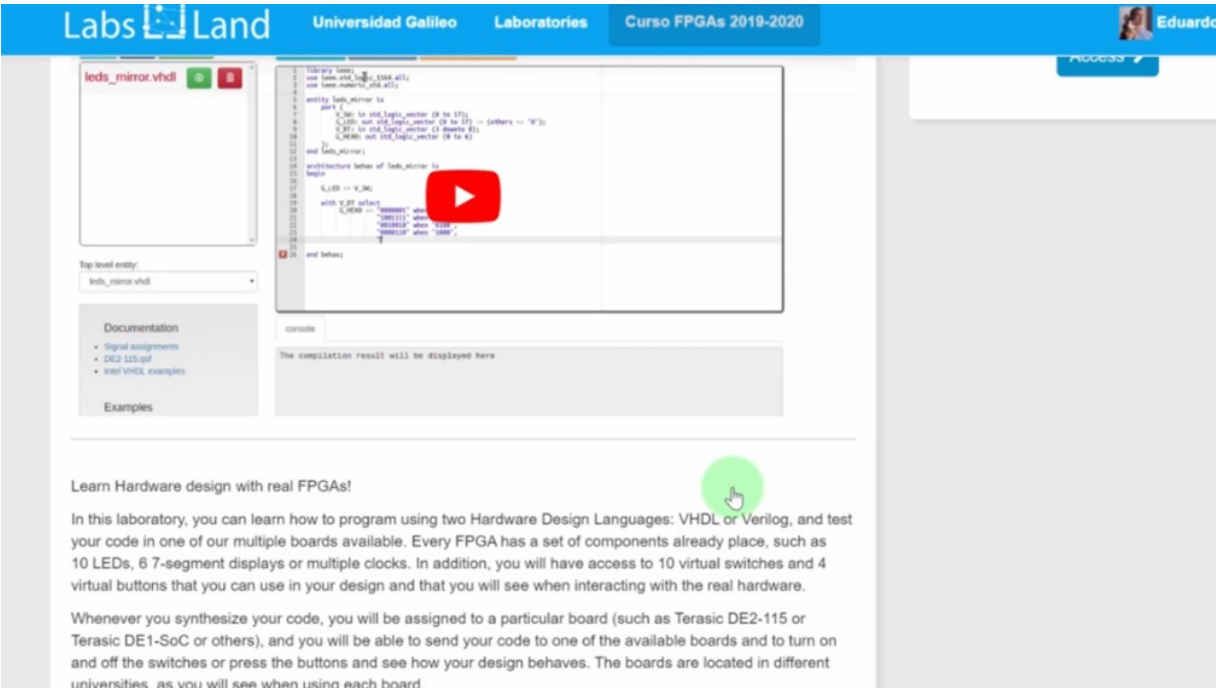


Serial Monitor

Send:

I am using a robot that is at some university at some real lab and I have one minute one and a half minutes so let's copy one program. Maybe this one avoid crashing. So I have downloaded the program into the robot so there it is. There you have it. The robot is really onsite avoiding a crash. Now let's run the line follower for that Roebuck so I will go again to the lab and you should know that I am getting access to that real robot so no one else is accessing it. If someone else from some other part of the world is trying to access the robot then AQ will form and I will have to wait in line. So let's load the line follower. These are examples of course but you get to write your own code. Right now it's sending the program to the robot and there it is. There you have it. The robot is performing a line following and he was calibrating first. Here you have the serial Monitor reporting what's going on on the robot. And as you can see it's following the line. Here are some buttons you can press here. It's working at maximum speed medium speed and slow speed. Now what we are most interested in are the FPGA labs. So let me show you the labs I have access to. So here we have an Arduino board lab the same robot we just saw. And we do have some electronics labs also. But here are some FPGA project labs. Here we have the FPGA laboratory which has several labs for FPGA development. We have

one on the Intel D2 115 board and another on the intel the E1 SLC board. Let's go to the FPGA lab and we have a very low I.D.. We could access sample and you get a tutorial which tells you which elements you can use. I'll just click next on the whole tutorial and here we have the LCD mirror that V file very low source code which is reproducing what's on the switches to the LCD. So let's download this code. We first have to synthesize and you'll see that it's using quartz tools. So we get a hint here that the target board will be an Intel board now that the synthesis process was successful. Let me just upload it into the device. The platform is offering an ultra FPGA. This is the D1 board. So we'll upload it there and once again it's fascinating to me that we don't know where in the world this is working right now.

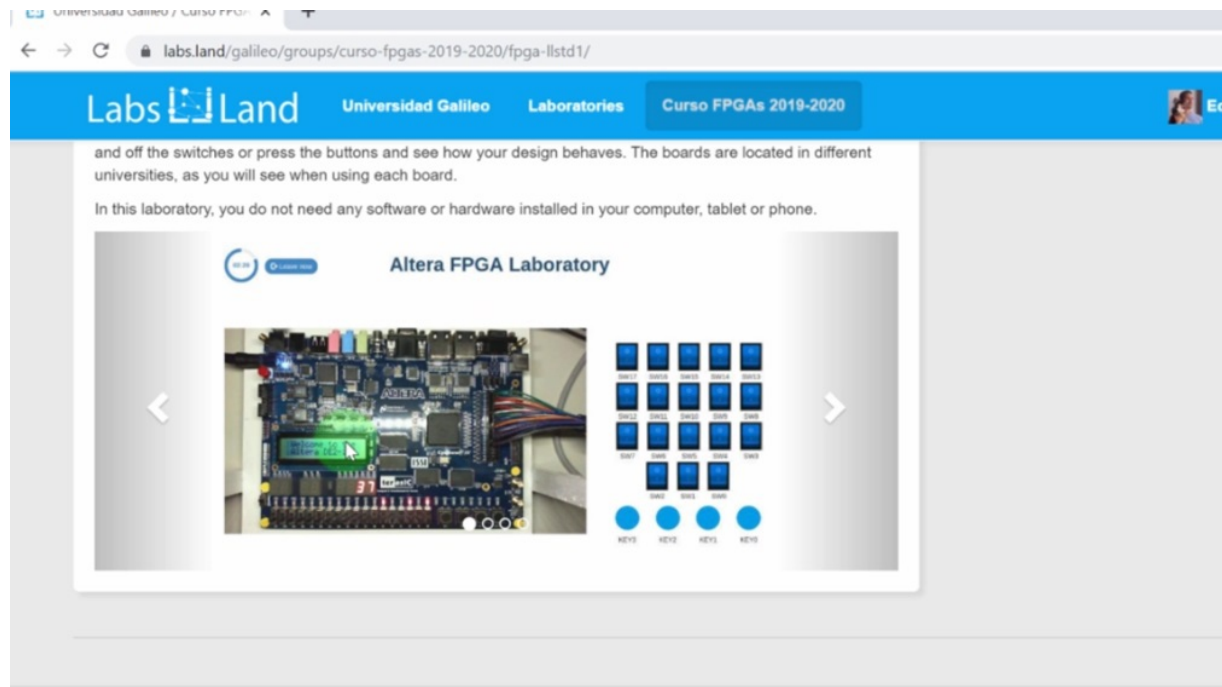


This is working at UPMC. This is a university in Spain. We have one minute and 46 seconds approximately. So your FPGA I.D. program I can download it into the board and you can see here that the board is being recorded and broadcast by a camera. And here it is. Here you have it. So the switches are right here at the bottom and a very complicated challenge is to get access remotely to these input

elements of the board as opposed to having access to the output elements because you only have to record with a camera. But here are the switches. So let's say I turn on the first switch and you can see that the matching LCD is on. Let me turn on the fourth switch which is this one and the LCD turns on. Let me turn off both and both LCD will turn off. Let me press a button. See what happens. Oh the button gets displayed on the segment display. So here's three Here's 0 and time is almost up. As you can see two minutes is more than enough to test your applications. And this is one of the boards available at the lab at UPMC. So there you have it. I hope you got a nice taste of what lab sent. Looks like.

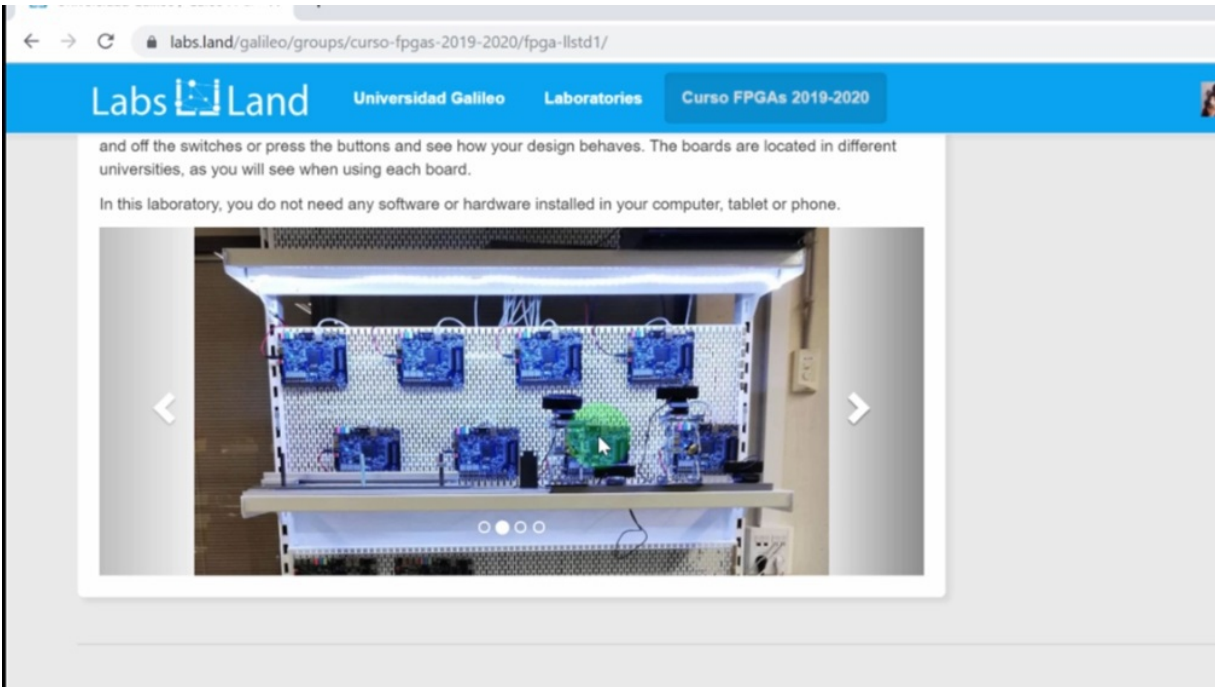
FPGA LAB

So let me tell you about the FPGA laboratory in this space. We have two links one for the VHA deal I.T. and one for the very long ITV. We will use very low because that's my language of choice but you could do it in VHF deal if you want so here we have a video where you get introduced to the platform and I recommend that you take a look at it. If you want and here we have some pictures. First we have the interface you'll get where you get to manipulate switches and buttons in the board.

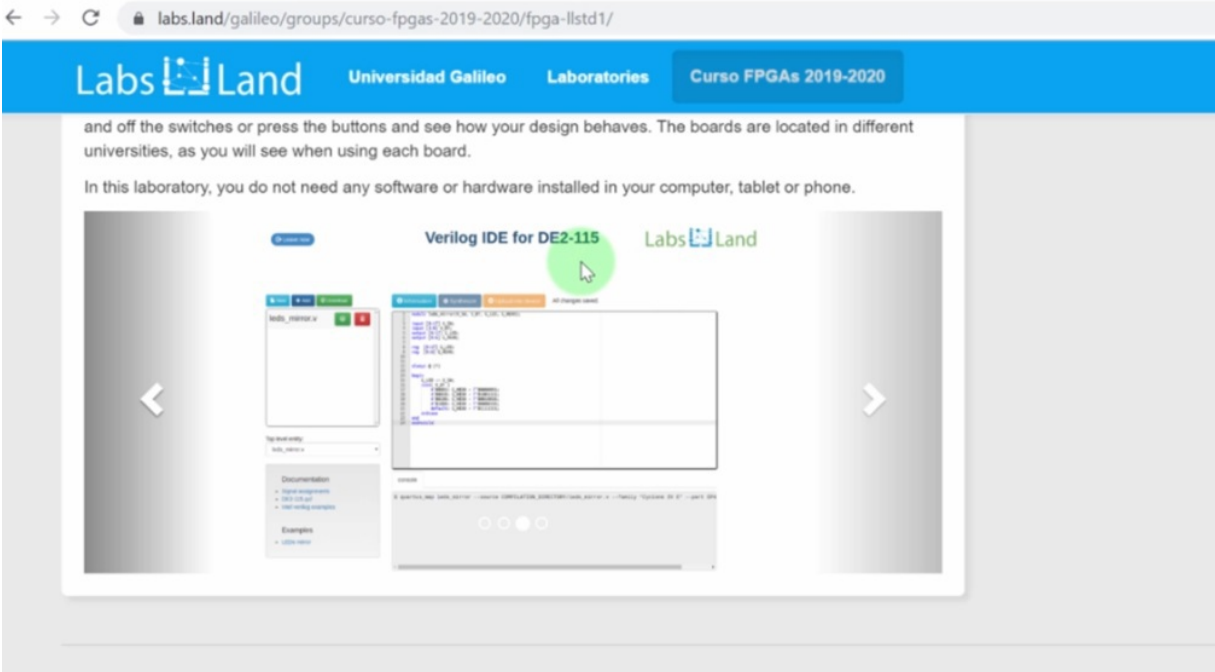


And this is actual footage of the board which is recorded and broadcast by a camera so that you can see what's going on and act on the board. Here is a nice picture of the boards onsite at some university or some lab where the people right there have mounted several boards not all boards have to be the same type and you're

not guaranteed to get the same exact board each time you reserved the labs.



So here we have some cameras that are pointing to specific boards which are assigned to you by the software every time you need to use this lab here's a picture of the I.D.. This is the very low I.D. and I recommend that you use your own files in your own computer and just copy and paste them here because the interface is not so very well organized in files but that's the way I've been using it. And this is the V HDL I.D. nothing new.



So notice that in this V HDL I.D. You have the de to 115 bought and we have the same one here for the very long 80 but in the demo maybe we'll get a different board and that's fine and we even may get a board from a computer that's in one part of the world at some university in I don't know the United States. And sometimes you may get one from a company that's in Spain so we are ready to take a look at the very low 80.

VERILOG IDE DEMO_PIN ASSIGNMENT

Now let's take a look at the very low 80s. And an example project I have just entered here. Let me start by telling you that this time we will be using the D E one S O C board because that's what was assigned to me. But let me refresh and hopefully I'll get a different board. Yes this is the the E2 115 board.



And as I refresh you'll see that it changes from time to time between boards. I think there are just two available types of boards right now. The E one isn't C and that the E to 115 but this gives us a nice opportunity to implement some compatibility between these boards and that's what you get if you scroll down a little in this documentation part. This link that says signal assignments has a

nice tutorial of the inputs and outputs you get in either board. So here we have an array that is called V S W virtual switches BBT virtual buttons.

The screenshot shows a documentation window with the following content:

Documentation - Signal assignments

LabsLand provides the following set of virtual inputs that you will see when you access the laboratory:

| Name | Description |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| input wire [9:0] V_SW | 10 virtual switches, numerated 0 to 9. You will be able to set them to 1 or 0 until you change the switch value. |
| input wire [3:0] V_BT | 4 virtual buttons, numerated 0 to 3. By default they will be 0, and whenever you click them they will become 1 for a short period of time. You can keep the button pressed if you want to keep sending a 1, but once you release it, it will become 0 again. |

Additionally, you will be able to refer to board peripherals with the following abstract names (which will be the same across boards):

| Name | Description |
|-------------------------|----------------------------------------------------------------------|
| input wire G_CLOCK_50 | Clock that works at 50 MHz. |
| input wire [9:0] G_LED0 | 10 red LEDs, numerated 0 to 9. They are guaranteed to be red. |
| input wire [9:0] G_LED | 10 LEDs, numerated 0 to 9. They are the same LEDs as the ones above. |
| input wire [6:0] G_HEX0 | Seven-segment display number 0. |
| input wire [6:0] G_HEX1 | Seven-segment display number 1. |
| input wire [6:0] G_HEX2 | Seven-segment display number 2. |
| input wire [6:0] G_HEX3 | Seven-segment display number 3. |

So these are 10 switches and these are four buttons and for the inputs we have the clock in the board which is a 50 megahertz clock. We have an array of LCD and we have a bunch of hex or rather seven segment displays. These are all separate in Intel boards. So you don't have to do any sweep or raster on them and you manage them separately. Those are seven bit arrays so you only send the active zero values for the segments and that's it so since all of these elements are common in the available boards you get to use these virtual elements and you just get to program them in your modules.

VERILOG IDE DEMO_

ADDER CODE

So in here I am using the same LCD mirror that V file that was available originally and I even left the same name here. This is not a very good practice but I'm just showing you how to do something on the board or even how to modify the LCD demo and I changed this to an other circuit. Let me change it to a multiplayer circuit to show you how easily this can be done. So it says that it shows on the display the addition. This will be the multiplication of the switches. So here we have all of the ports virtual switch and you're only required to enter the input output.

The screenshot displays the Verilog IDE interface. The top navigation bar includes buttons for 'New', 'Add', 'Download', 'Information', 'Synthesize', and 'Upload into device'. The main workspace is divided into several sections:

- File Explorer:** Shows two files: 'hex2_7seg.v' and 'leds_mirror.v', each with a green play button and a red delete button.
- Top level entity:** A dropdown menu currently set to 'leds_mirror.v'.
- Code Editor:** Contains the following Verilog code:

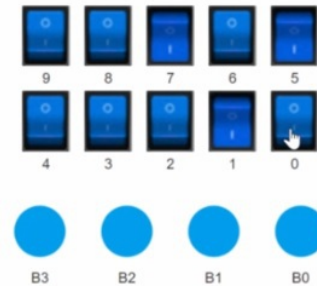
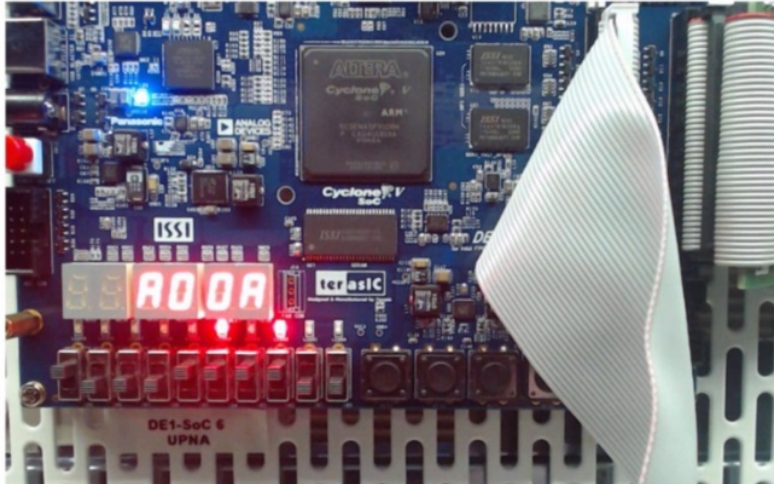
```
1 //
2 // Multiplier circuit
3 //
4 // Shows on the displays the multiplication of the switches.
5 //
6 module leds_mirror(V_SW, G_LED, G_HEX0, G_HEX1, G_HEX2, G_HEX3);
7 input [9:0] V_SW;
8 output [0:9] G_LED;
9 output [0:6] G_HEX0;
10 output [0:6] G_HEX1;
11 output [0:6] G_HEX2;
12 output [0:6] G_HEX3;
13
14 reg [0:9] G_LED;
15 reg [7:0] RES;
16 wire dp0, dp1, dp2, dp3;
17
18
19 hex2_7seg hex0(RES[3],RES[2],RES[1],RES[0],
20 G_HEX0[0],G_HEX0[1],G_HEX0[2],G_HEX0[3],G_HEX0[4],G_HEX0[5],G_HEX0[6],
21 dp0);
22 hex2_7seg hex1(RES[7],RES[6],RES[5],RES[4],
23 G_HEX1[0],G_HEX1[1],G_HEX1[2],G_HEX1[3],G_HEX1[4],G_HEX1[5],G_HEX1[6],
24 dp0);
25 hex2_7seg hex2(V_SW[3],V_SW[2],V_SW[1],V_SW[0],
26 G_HEX2[0],G_HEX2[1],G_HEX2[2],G_HEX2[3],G_HEX2[4],G_HEX2[5],G_HEX2[6],
27 dp0);
```
- Documentation:** Lists 'Signal assignments' and 'Intel Verilog examples'.
- Examples:** Lists 'LEDs mirror'.
- Console:** Shows the synthesis status: 'Analysis & Synthesis Status : Successful - Mon Nov 25 20:02:31 2019'. It also lists 'Quartus Prime Version : 17.1.0 Build 590 10/25/2018', 'Revision Name : leds_mirror', 'Top-level Entity Name : leds_mirror', 'Family : Cyclone V', 'Logic utilization (in ALMs) : N/A', and 'Total registers : 0'.

You're going to use. So notice that I'm using the switches right here. We are not going to need the virtual buttons. So I will get rid of them

and notice that this is the very log 95 syntax so that in the ports list you specify the names but later you have to specify whether they are inputs or outputs and whether they are registers or not. So I got rid of the buttons which I'm not going to use and I'm going to use the segment displays hex 0 through hex 3. Those are for displays and the LCD arrays now notice that for some reason this code has the range specified from the least significant bit to the most significant bit. This may be a problem but I have only reversed the order for the switches because of the way I'm using the code later on. Now look at line 14. Here we are specifying that the LCD array is a register because we are going to load a value into it with an assignment and I have an 8 bit Register called rests for result. So this result is going to go to the two rightmost LCD displays and the two left most segment this place will show what's on the switches. So here are my Seven segment displays. You may want to hit boss and verified that this is all correct but what I'm doing here is that I am instantiating hex to seven segment decoders. So what are these. These are called hex 0 through 3 and these modules are specified in this other module. I wrote sometime back and this is a hex to 7 segment decoder. It has too many ports but anyway they are four bits for the inputs and 8 bits for the outputs in the inputs. D is the most significant bit and a is the least significant of the single nibble that goes in and the outputs are segments A through G. And additionally I have the decimal point here which I am not using here at the end you can see that I'm assigning one to it and these are all active 0. Now for the rest of the lines of code you'll see that I am assigning the Boolean algebra a representation of the function for each of these segments.


[Leave now](#)

Altera FPGA Laboratory

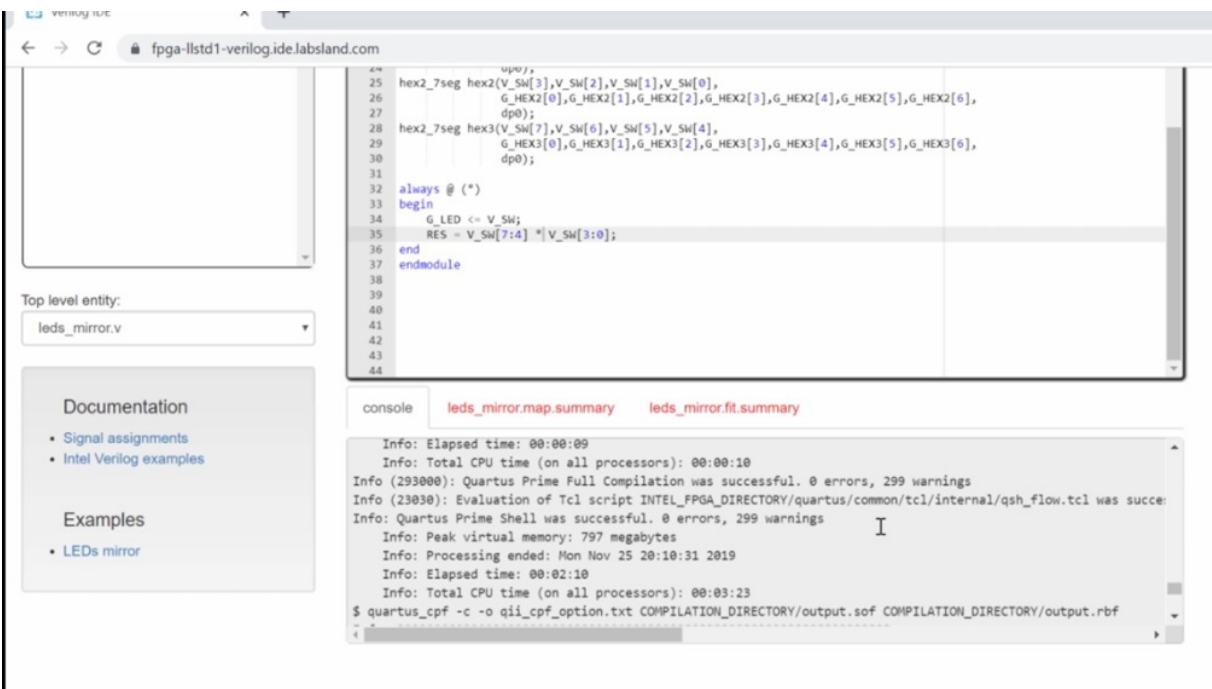


And notice that I'm negating it with the not operator at the beginning because what I'm doing here is the sum of minterms. That's a form of expressing boolean functions and what it says here is that segment 8 will light up whenever the input is 0 2 3 5 6 7 8 9 A C E or F you may check the code for the rest of the segments and you will see that this is just a hex to 7 segment decoder. Anyway going back to the LCD mirror file which has anything but that it has my multiplication demo you'll see that I'm instantiating all of these decoders. So notice that the rightmost one which is hex 0 takes its inputs from the lower nibble of the result and the higher nibble of the result is sent to the next hex display. So this is how we print to the right the result and what goes out the left are the two upper end. So we have switches three through zero the right move switches to them so to speak. Second hex display and the leftmost gets the other upper end from switches 7 through for the leftmost switches. Now the call is very very simple. It's an always block which always executes this part right here which are only two lines. First I'm assigning to the LCD exactly what's on the virtual switches so you'll see the LCD light up with the switches. That's the original demo but I'm also assigning two risks the addition right now between the

switches 7 through 4 which are the four switches at the left all of this operated with switches three through zero which are the right most.

VERILOG IDE DEMO_ ADDER LIVE DEMO

So we are now ready to synthesise. So let me just press this button right here and the synthesis process starts and you can see it right here that the compilation has started and this will take a while here at the bottom we can see the results from the compiler tool chain in a console window.



So if we scroll down we'll see all of the warnings and information messages and it's still working. By the way we also get some files right here from the different steps of the implementation process for example this one says LCD mirror map summary for the map step. So this is the result from that step let's go back to the console and we have more results so after a while we get that the program was

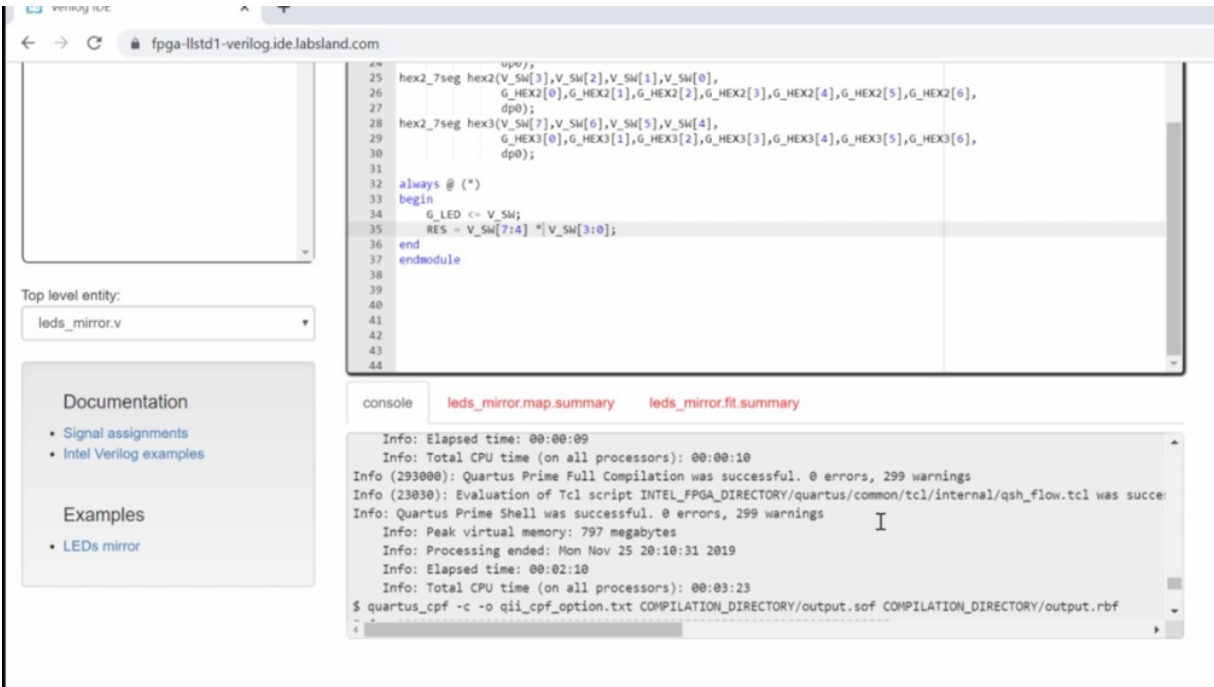
successfully verified and compiled. So take a look at the console and at the very end we get a message that says info. Prime convert programming file was successful zero errors zero warnings. So that's what we were looking for.



And now we can upload into the device as you can see I am being offered an FPGA D1 board which is neither of the two we saw before. I'll upload now as you can see it says reserving I am next in line. That's good. Great. So this is where we get to download our code into the board. This is the D1 board and I will program it. Notice that I only have two minutes right here. But that's more than enough. Once I program it you can see my displays showing I think it was an either the way I left it. So let me put one one that will be 10 plus zero equals zero eight 10 plus three equals a plus three equals the more for example a plus a must equal 20. Which is 14 in hexadecimal.

VERILOG IDE DEMO_ MULTIPLIER LIVE DEMO


Let's go back and change this to a multiplier because as you saw this was supposed to be a multiplier and they just left this operator right here as an addition. So let me change that to an asterisk and notice that I'm being offered a d e one S O Seaboard now. So now let's see it working as a multiplier. Let me synthesise and after a while we get it to compile again. And let me upload into the device once more I get a D E one board. Now let me download this program into my board and notice how this is different. Now let me multiply 10 times zero equals zero and ten times let's say two equals 20 which is fourteen in hexadecimal ten times three equals thirty which is one e in hexadecimal because this is sixteen close fourteen that's thirty. So there you have it and noticed that the timer is at one minute. So again this is two minutes is more than enough for testing your simple designs.



And if you need more time you can always reserve two more minutes. So let me show you how we can reserve another board. Notice that it says upload into your device. So I need to compile and verify I'm not allowed to upload again but that doesn't mean that you'll have to wait about five minutes for it to compile. You can synthesize but this step checks for a cache where if there are no changes in your code then the software knows that synthesis was previously done and so you'll get this right away. Let's upload again. We are next. Yay and once more we get to programmed FPGA board and you'll see exactly what we expected. So here we have four times two equals eight. And there are a couple of things I want to see. First this lab is located at Spain and it's from the UDP in a university. And we might have gotten another board from another site. And the other thing is that this board is being controlled by these virtual switches we have here. And as you know those switches are actually shown here on the board as you can see this LCD and this LCD are showing the state of my virtual switches but these are not the physical switches on the board. So what's going on here is that there is support with all of the input output being overridden by our application.

RECOMPILING

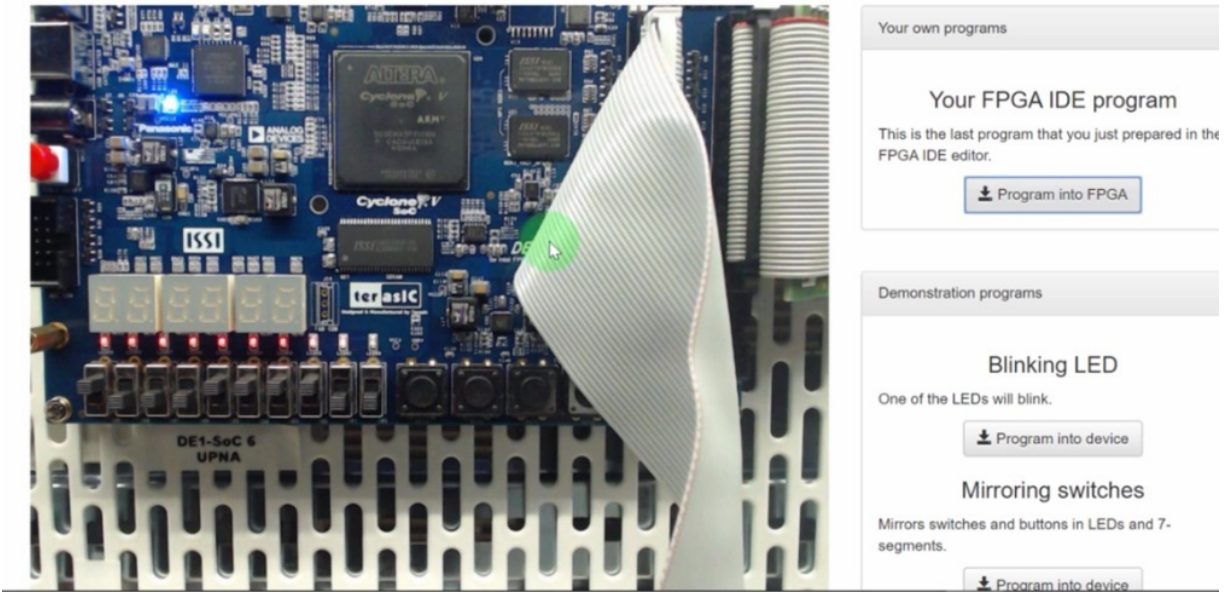
So finally let me show you how the cash works. So if you synthesize your design labs and keeps a cache of all of these produced files and if nothing has changed in your source code then it will use those files so you won't have to wait for it to recompile.



The screenshot displays the Verilog IDE interface. At the top center, it says "Verilog IDE (DE1-SoC)". To the right is the "Labs La" logo. A green notification box in the center states: "The program was successfully verified and compiled (10:54:59 AM)". Below this, there are buttons for "New", "Add", and "Download" on the left, and "Information", "Synthesize", and "Upload into device" on the right, with the text "All changes saved." to the right of the buttons. The main area is a code editor showing Verilog code for a 7-segment display driver. The code includes declarations for registers and wires, and instances of a "hex2_7seg" module. A status bar at the bottom left indicates "Top level entity:".

```
14 reg [0:9] G_LED;
15 reg [7:0] RES;
16 wire dp0, dp1, dp2, dp3;
17
18
19 hex2_7seg hex0(RES[3],RES[2],RES[1],RES[0],
20 G_HEX0[0],G_HEX0[1],G_HEX0[2],G_HEX0[3],G_HEX0[4],G_HEX0[5],G_HEX0[6],
21 dp0);
22 hex2_7seg hex1(RES[7],RES[6],RES[5],RES[4],
23 G_HEX1[0],G_HEX1[1],G_HEX1[2],G_HEX1[3],G_HEX1[4],G_HEX1[5],G_HEX1[6],
24 dp0);
25 hex2_7seg hex2(V_SW[3],V_SW[2],V_SW[1],V_SW[0],
26 G_HEX2[0],G_HEX2[1],G_HEX2[2],G_HEX2[3],G_HEX2[4],G_HEX2[5],G_HEX2[6],
27 dp0);
28 hex2_7seg hex3(V_SW[7],V_SW[6],V_SW[5],V_SW[4],
29 G_HEX3[0],G_HEX3[1],G_HEX3[2],G_HEX3[3],G_HEX3[4],G_HEX3[5],G_HEX3[6],
30 dp0);
31
32 always @ (*)
33 begin
34 G_LED <= V_SW;
35 RES = V_SW[7:4] * V_SW[3:0];
36 end
```

So let's synthesize this again and as you can see it was already compiled so it used the Cash version. Now let me upload it into the device.



Now I'm being offered an FPGA D one let's upload again reserving I'm next in line. Yay and here we have the FPGA so it can run our multiplayer once more. Three times three equals nine.

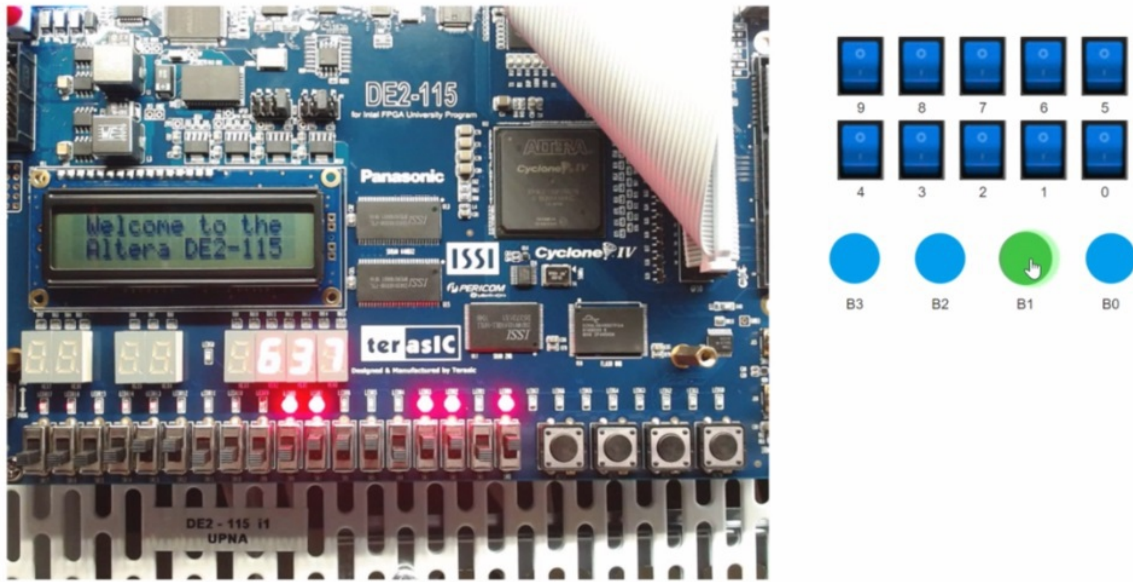
UPLOADING DEMOS

Finally I want to show you how you can install example applications into the FPGA. Notice that this time I'm being offered an FPGA D2 115 board so here we have the button to download our current design but we can download some other example programs.



The screenshot shows the Altera FPGA Laboratory interface. At the top, there is a timer at 04:46 and a 'Leave now' button. The main header reads 'Altera FPGA Laboratory'. Below this, there is a photograph of an Altera DE2-115 FPGA development board. The board features a Panasonic LCD displaying 'Welcome to the Altera DE2-115', several 7-segment displays showing '8.8', and various components like the Cyclone IV FPGA, ISSI memory, and ternaC components. To the right of the board image is a software interface with two sections: 'Your own programs' and 'Demonstration programs'. Under 'Your own programs', there is a button labeled 'Program into FPGA' with a green circle highlighting it. Under 'Demonstration programs', there are two options: 'Blinking LED' with a 'Program into device' button, and 'Mirroring switches'.

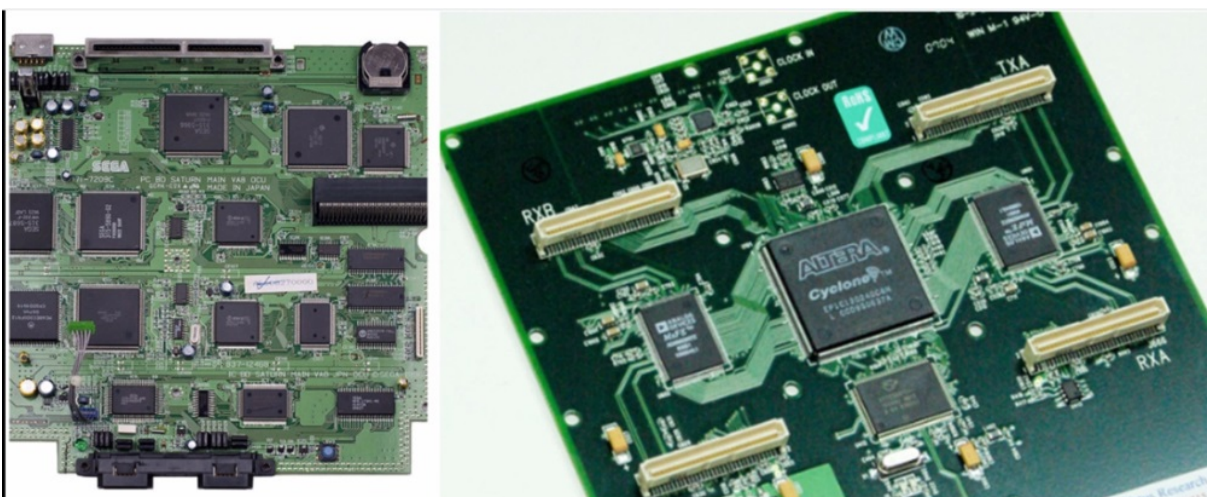
We have the blinking LCD right here. We have mirroring switches and we have a clock. So let's install the clock. So as you can see here we have a counter going on. This is counting on the LCD. It says well 13 14 and you can see the count in the displays. Hopefully let's see what the buttons do. We don't have access to the source code so as you can see the clock is running faster when a press B three.



Let's see what happens when they press B to OK. So these seem to be different speeds for the color yes we still have three and half minutes with this board. Let's see what the switches do. If they do anything yes switch 0 resets the count and inhibits the count. Let's see switch one apparently all switches do that. It's straight for a switch. Eight yes. So there you have it. You get to download example applications into the FBI s.

MOTIVATION_ HARDWARE DESIGN

One motivation example they want to give you before we start talking about FPGA. Is the purpose of using an FPGA. As you may already know FBA have grown in popularity over the years and now they are use in virtually every serious mother in design because of their convenience. And one of these advantages is requiring less chips in a board. By reducing the chip count you can reduce the price and the energy consumption of your system. So let me give you a brief example of that. In this illustration we have the motherboard for the Sega Saturn video game console which goes back to the 1980s. And as you can see you only have to take a look at the chips right there and it's definitely a board filled with chips and looking at that board. Today I think it would be reasonable to have at least some of those chips integrated into one single chip that could implement their electronics.

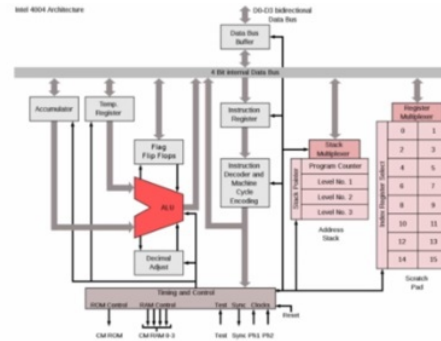
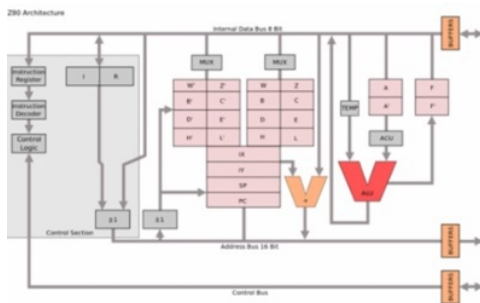


Motivation: Hardware Design

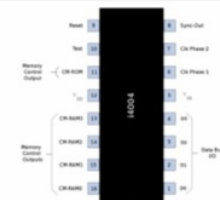
So the chip count could drop to say four or five chips instead of the 20 or so you can see in that illustration. And now let's take a look at a more modern board which is a U.S. R.P.. This is a general purpose radio controller to implement several wireless systems. Among those we have cellular phones and Wi-Fi devices. And as you can see that board has in the middle there a chip that's a cycling FPGA just compared the chip count between these two boards. I know they are not the same thing. But trust me the computational power of the board at the right which definitely was designed later than 2010. Well that board is computationally much much more powerful than the Sega Saturn game console from the 90s. So with BGH have been one of the technologies responsible for allowing electronics to become smaller and smaller FPGA are one of the reasons why your mobile device fits in your pocket. So once again a very nice motivation is that if VGX today are implemented inside chips that have other electronics inside. And the FBI part is included to implement whatever you want in that circuitry so that your electronics your external electronics can be well fewer and fewer.

MOTIVATION_SOFT PROCESSORS

Another motivation I'd like to remind you about is making us keep you in the third project of this series. We will do this and a you can be quite simple here we have at the left a chip from Zilog This is the Ziad. This is CPQ. That was very popular in the 80s and as you can see those blocks that appear there are simple digital circuits not so different from the very long modules we have already designed. So we will do something like that. And at the right we have the first microprocessor by Intel not the first ever.



Motivation: Let's make a CPU!



That was by Texas Instruments. But really the second microprocessor ever and that's 40 all four. And again they just want you to pay attention to how many blocks it has. This is a very simple

computer. So we'll do something similar to these in the next course. But let me tell you that the use of soft processors aren't really considered wonderful designs today. They are actually widely used in FPGA. That's why vendors like Altera and Xilinx provide their own designs for their soft processors and there are tons of soft processors out there. So this is a nice motivation especially if you are keen on computer architecture.

INTRO TO FPGAS

Now let's get to know field programmable gate or race or FPGA. And that's exactly what we will do in this project. We'll answer several questions. So show us what exactly is and if BDA how they are implemented. Who makes them. What can they do with them. How can a program them and so on and so forth.

In this lesson: Let's get to know FPGAs

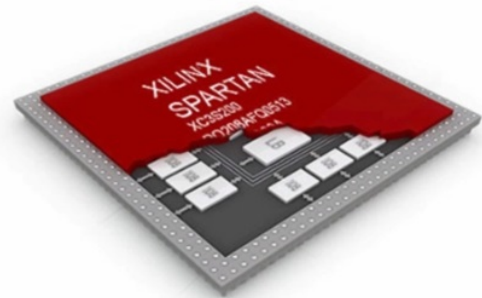
- What exactly is an FPGA?
- How are they implemented?
- Who makes them?
- What can I do with them?
- How can I program them?

So let's get started. So for the first question what exactly is an FPGA. Well and the FPGA is a gate or a type of integrated circuit. That means that it's a very specialized type of chip that includes a whole lot of gates inside organized as some sort of re But there are many many many gates. I'm talking hundreds of thousands if not millions of gates. And the fun part is that it's field programmable. And what that means is that the user can program it you as an engineer

get to program what is going to be implemented with those many available gates.

What exactly is an FPGA?

- It's a Gate-Array type of Integrated Circuit.
- The fun part: It's Field-Programmable!



So I like to think of it Biggie's as shapeshifter integrated circuits and that's because they can become whichever digital system you would like them to become if you want a counter you can get a counter if you want a digital clock or maybe you want a serial port controller or a very specific state machine. Well you can have any digital device you'd like. So this chip will become whatever you want it to become.

FPGA OVERVIEW

So let me give you an overview about field programmable gate arrays so they contain logical cells which are special blocks of logic devices that include lookup tables flip flops others and some other digital devices inside. Remember you decide how you want them to behave and so you enter such description using one of several methods that are today available.

Field Programmable Gate Arrays

- They contain logical cells which include Lookup Tables, FlipFlops and Adders.
- You decide how you want them to behave and enter such description using one of several methods.
- A very popular method is using Hardware Description Languages.
- A synthesis tool produces a *bitstream* file that sets the connections between cells when downloaded into an FPGA.

The most popular by far is a hardware description language such as very long or VHDL and to program them you need a synthesis tool that produces finally a binary file which is a bitstream file. And what this binary file contains is all of the connections that are required inside that FPGA to implement whatever hardware you have described in your very long or VHDL source files.

FPGAS VS ASIC

Let me show you some highlights when we compare FPGA to traditional integrated circuit so traditional bases are known as ASAC that stands for application specific integrated circuit and as the name already says they are designed with a very specific purpose.

FPGAs compared to Traditional ICs

- ASIC: Application Specific Integrated Circuit.
- As the name suggests, they're designed with a specific purpose.
- An FPGA can be programmed by the (engineering) user.

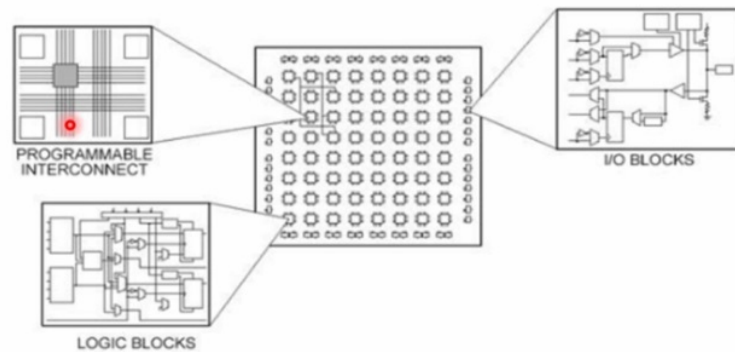
I'm talking about a multiplexer maybe an integrated circuit that contains some gates inside or maybe counter a shift register or even a microprocessor. All of those are application specific and those async well enough PDA can be programmed by you by the user and by user I mean an engineer the person who's in charge of making all of the hardware being implemented inside that FPGA. The whole objective of using an FPGA remember is to implement all the logic you want to go into your board.

WHAT'S INSIDE AN FPGA

So what's inside an FPGA and if it consists of a lot of special blocks but in the simplest sense it consists of three main parts which are interconnects or enter connectors which are the conductors that will interconnect all of the hardware and you will design.

What's inside an FPGA?

- Interconnectors
- Logic Blocks
- I/O Blocks

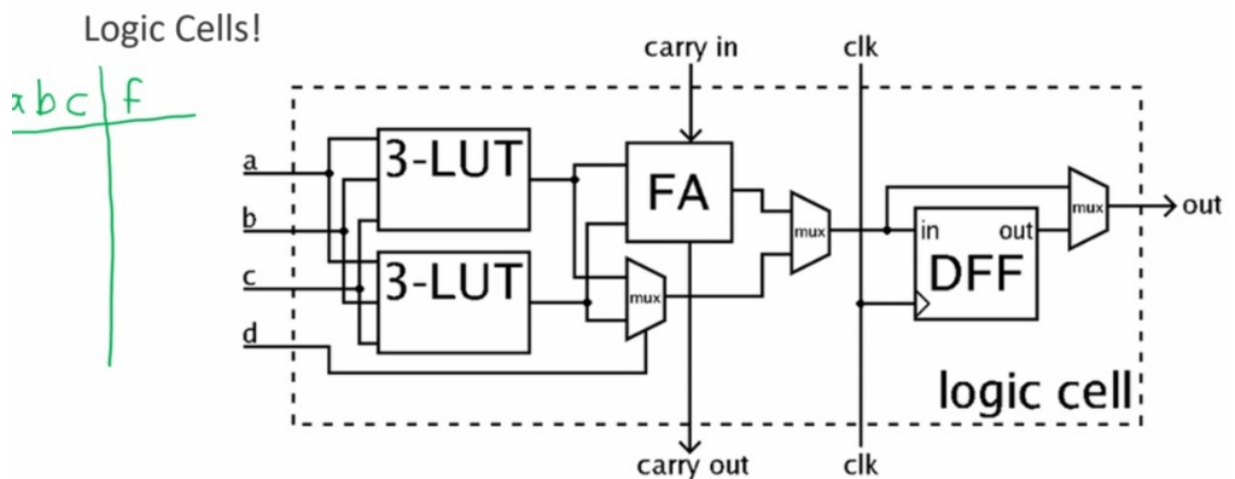


Next we have logic blocks which implement all of that logic. This is the main bulk part of the FDA. And finally we have input output blocks which go to the bins to the actual input output pins. Let me tell you more about all of these parts.

WHAT'S INSIDE LOGIC BLOCKS - LOOKUP TABLES

So inside a logic block we have logic cells. And what's a logic cell. It looks like this. So let me walk you through the elements of this example logic cell and be aware that not all logic cells and most likely no logic cell is exactly like this one. This is just an example. But they contain several elements similar to this. First we have a section with lookup tables. That's what L stands for.

What's inside a Logic Block?



And as you may know a lookup table is an implementation of a function that consists of just a table you fill up the table as a memory and you look up its values by entering addresses into it or input data

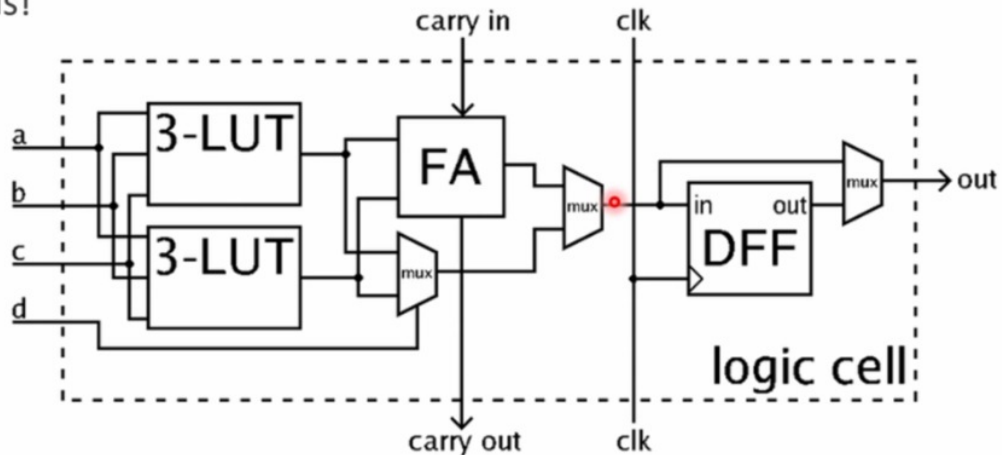
to it and it will spit out whatever that function is supposed to send in itself. But because it's a function and so we have a three lot here what it means is the implementation of whatever function that has three inputs which are A B and C does this one function one three input function that goes to this node. Next we have another three locked connected to the three same inputs which go to this node right here. And so you can do whatever you want with these two functions. You have a function of A B and C here. And we have another function of Fabian see here. If you want to make it a forward lot you can do it because you have another input here D which may act if you want as a multiplexer for either using this function the one at the top or the function at the bottom in this output line right here. So for example one thing this lookup table may implement is something like this. We have a truth table right here. We have a function that's call f one and another one that's called F 2. So let's say we have all of the combinations right here. And so you may implement whatever you want for EF 1. Let's say this function. I'm just making it up and whatever you want Forth to say this. So the top lookup table may output F 1 and the bottom lookup table may hold. But as to no there must be a way to enter these truths fables into the lookup tables for each logic cell and we'll talk about that in a bit. So let's move on.

WHAT'S INSIDE LOGIC BLOCKS_ADDERS AND FLIP FLOPS

So you have the option of performing any combinational function either of three inputs or four inputs. If we combine the two lookup tables with this multiplexer right here and so we would have that for input function right here or just a three input function right here. Now if you want to do arithmetic with your digital designs you have a very small element of arithmetic right here which you may already know. This is a full either. So if you remember a full leather has three inputs one input for one of the operant. The next is the other upper end and it has that carry in input. And so this full either will send it some right here in this output and it's carry output will go outside of the cell right here no notice how a multiplexer is able to select what to send to the next stage.

What's inside a Logic Block?

Logic Cells!



So we could simply send the combinational part we got from the lookup tables through this first multiplexer right through this second multiplexer into the next stage without using the full either or we might use the full leather by selecting with this multiplexer to send this part this input to its output. So there is some line that controls this multiplexer. We'll talk about this a bit later. Next with this signal which may be an arithmetic calculation or an arbitrary combinational function we can do either a sequential function that is by using the smallest sequential element out there which is a flip flop. This is a flip flop and we can use this logical signal as its input and later use its output as the actual output of this logic cell. We can choose to use this output by means of this multiplexer right here. And if we don't want the sequential functionality of this The Flip-Flop we can just choose to not use it by selecting with this multiplexer to use this input to appear on its output. Notice that since this is a flip flop it requires say clock signal which is a bus available through out the logical cell's. Now let's recap. So logic cell allows you to do whatever combinational behavior you want by means of a couple of lookup tables. You may also perform very simple arithmetic with a full other which is connected among logic cells and you can make it sequential if you want with a flip flop. Remember one flip flop gives you one

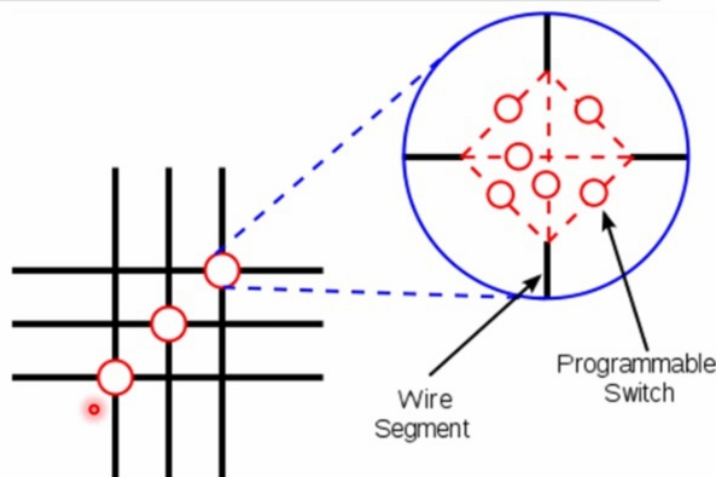
variable in your sequential system the clock signal traverses all of those logic cells. And so you get a very nice flexibility to perform either of these three functionalities or any of their combinations. It all comes down to what you do with this multiplexer this one and this one again. There are a lot of variations of these logic cells but the same principle of having a very flexible design is always there.

WHAT'S INSIDE INTERCONNECTS

Know what's inside the interconnects or the interconnecting hardware well in there we have programmable switches. And by that I mean analog switches or transmission gates which are switches implemented electronically by means of transistors namely mosfet transistors and so you have a network of horizontal and vertical lines where you have all of your circuitry and these lines interconnect the many elements of the logic cells. So I'm talking about the inputs ABC and the.

What's inside Interconnects?

Programmable Switches!



In the previous example the carry input or the outputs the carry output the cell output or even a clock bus. And so you interconnect all of those by means of these vertical and horizontal lines in each of

these junctions we have between a horizontal line and a vertical line is composed of several single pole single throw switches which open or close. So you may perform whichever combination of open and close switches you want in here for example if you want to connect the left way to the top right here you can do it by crossing this diagonal top left switch. And if you want to connect these two lines you can also do that by closing the switch. So let's say you want to connect this line I'm showing you in green with this line right here. So you would only have to close this switch. Let's say you want to include let's say the line at the right with those two connections. You can do it by either closing this switch right here the top right or by closing the middle switch right here. And so you have a lot of options for this. And if you want to connect all of these lines you can close any of the remaining switches but you have a lot of flexibility I hope you can see that.

WHAT'S INSIDE I_O BLOCKS

So what's inside the IO blocks or input output blocks. Well we have amplifier's and attenuators that is because the inside of an integrated circuit especially a digital integrated grid is usually using lower power inside that means low currents and low voltages. It doesn't matter if your logic uses 3.3 volts on the outside or if it uses 5 volts on the outside.

What's inside I/O Blocks?

- Amplifiers and Attenuators.
- Level Detectors.
- Protection circuitry for ESD, overload, etc.

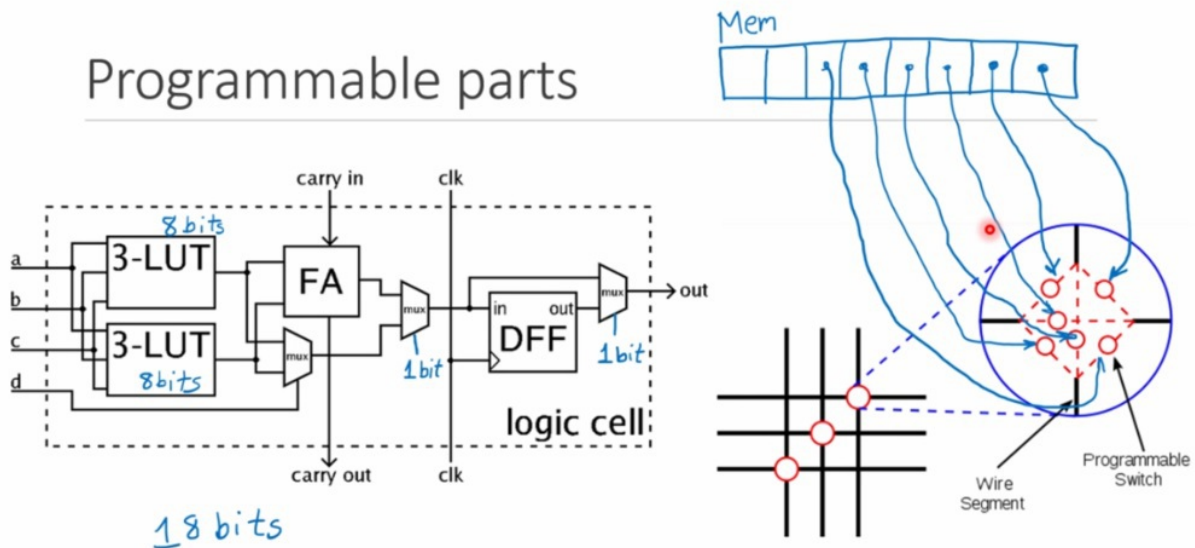
Chances are the implementation that insight implementation is going to have less voltage for the logical one. For example 1.8 volts and so it requires those levels changer's for the inputs it requires level detectors to determine what's the logical one and what's a logical zero or a high or low level. And they also have protection circuitry for

example for electrostatic discharges or to protect the outputs from overload and so on and so forth.

WHICH IS THE PROGRAMMABLE

So let's address something you may be wondering. Which is the programmable part. Remember we know that if BGH contain interconnections they contain logic blocks and input output blocks. So which of those elements is programmable the interconnections the logic blocks the input of the blocks Well the answer is none. There is a memory device somewhere that implements Well a big memory map and each of those bits implement whatever interconnection you want to make whatever lookup table you want to implement and whatever connections inside logic blocks you want. And so this affects the interconnections and it also affects the logic blocks and there are also some implications on the input output blocks. So let me tell you what I mean. So some of the programmable parts are the interconnections. And so you may have some bits in the memory I will show you a byte for now. Remember this byte belongs to the memory that implements the biggest connections. And so one bit right here goes to control all the time. Just one of the switches and other bit controls. Another switch. And so one so as you can imagine there are a lot of bits required to implement all of these interconnections. These are only the six bits required for this interconnection but this one requires also six bits. And this too. And what about the ones that aren't shown here for example. This one and this one and this one will all of those require six bit in this particular model. And I am showing you just one byte right here. Imagine how much memory is needed to implement say a hundred thousand interconnections. That's a lot of memory and the interconnections are not the only programmable parts.

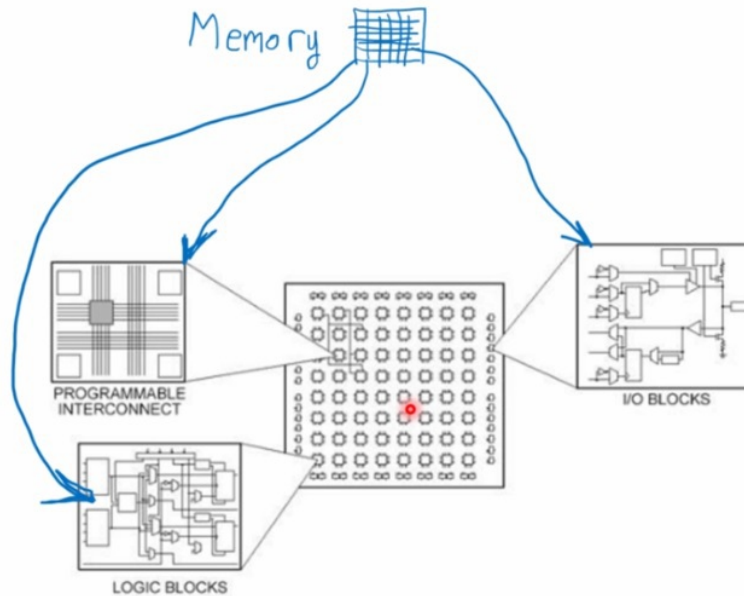
Programmable parts



We also have the logic cells which have a lookup table that has to be implemented somewhere. Remember these are functions functions that have a truth table so that table goes in here and guess where that table is stored. You guessed it in that memory. So more bits are required to implement each of these tables. That top table requires 8 bits. The bottom table requires also 8 bits. And what about the decision of what to do with this multiplexer. Well that requires one selection input as well as this multiplexer. So when this logic so we are talking about one more bit here and one more bit here. So overall it comes to 18 bits per logic scale and six bits per interconnection of four ways which like I said is adding up to a very large memory.

So...

Which is the programmable part?



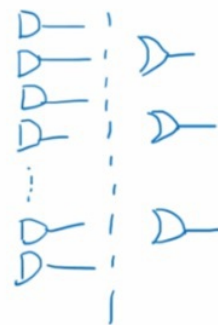
So to answer the question Which of these parts is programmable. Well this memory is programmable and this memory is controlling the interconnections the logic blocks. And like I said some input up of blocks can be affected by this memory. Let's say you want to configure a high power output or. Well these bins are input or output pins. So to configure them as inputs or outputs you need some device that will remember that direction of data. You can also set a open drains state for example to connect it to a bus and you want it to have the high impedance state. You can do that on the actual Beenz. And so this memory is what we are interested in and this memory will control how all of this available logic will behave. We need good software to perform this. We need software that will convert our hardware description language source files into the actual implementation of this memory that will make this hardware behave as we want. That's very advanced software indeed.

FPGAS VS CPLDS

Now among the gate array devices you may have heard of Sebelius which stands for complex programmable logic device Well these devices are part of the evolution of the race and their complexity is said to be between that of a people which is one of the most primitive programmable gate arrays and the complexity of NFPA and about details. Well speel these are said to be their evolution. So they have the traditional and or blocks. So by that I mean that we have a lot of and Gates inside connected to some level of switches. Where we may have some other levels of gates but eventually we get or Gates does this to implement what is known as a sum of products anyway.

FPGA vs. CPLDs

- CPLD: Complex Programmable Logic Device
- Their complexity is between PALs and FPGAs.
- Evolution of PALs: Traditional AND-OR blocks.
- FPGAs are much more widely used today.



We have lots of gates and lots of or gates. That's the main architecture of a. But we won't spend much time talking about

CBLDF because FPGA are much more widely used today. But here's a fun fact. Did you notice that FPGA don't contain an actual gate array but rather some more involved digital devices like lookup tables and multiplexers and others and flip flops. And so they are not exactly gate arrays as the name suggests. On the other hand we have CPLD these which are said to be complex logic programmable devices. And that sounds more like the FBI. So let me make the observation that the yeas are actually complex programmable logic devices while Seville these are field programmable gate arrays. Think about it for a minute.

HOW IS AN FPGA PROGRAMMED

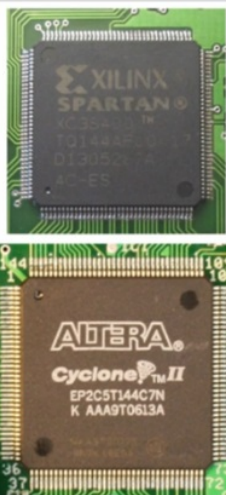
So how do you program an FPGA Well you enter your hardware into a software tool by using either schematics or you may also use equations. But the most popular is the hardware description language like very low or VHDL in the picture we have a screenshot of the Vogl which is a tool by site links so you compile your code just like programming code using a compiler well for FPGA so you use a development and synthesis tool which does pretty much that compile the code to produce the bitstream file that goes into the memory of the FPGA those tools maybe VI Vado Design Suite by Xilinx or quarta surprised by Intel or Altera.

- This code is compiled using a Development and Synthesis tool like Vivado Design Suite, by Xilinx, or Quartus Prime, by Intel.
- The output bitstream file is downloaded to the FPGA with a software tool.



Those are pretty much the two options you have if you are going to use the most popular FPGA Souder. And so the last step is to download that output file into an FPGA. You need a software tool for that. And this software too is often included in the design suites I just mentioned. So in the picture we have a software that is called adapt by a company called Digital int and the one shown is for the bases to board by vigilant. So each manufacturer offers different tools for their boards and we'll get to use a couple of those a bit later in the project. So how many times can a program an FPGA. Well that's an interesting question because it all depends on the technology.

How many times can I program an FPGA?



It depends on the Technology:

- SRAM (Indefinitely).
- Antifuse (Once).
- EPROM (Hundreds)
- EEPROM (Hundreds of Thousands).
- Flash (Millions).
- Fuse (Once).

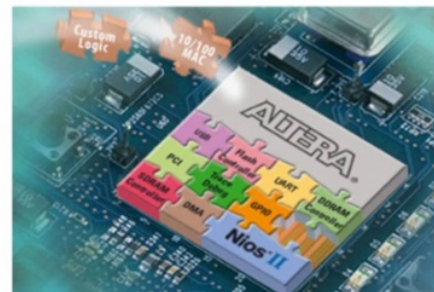
The memory is implemented in remember we use a memory inside the FBD to store its behavior and that memory contains all of the interconnections and the whole behavior of your design. And so that memory can be implemented in several ways. The most popular is an SRAM memory that is a static ram memory which is the fastest and the most expensive memory cell there that one can be program indefinitely because it's made out of gates. There are some other technologies for example. And if use can be as the name suggests only programmed once we have the classical eeprom memory which can be erased but not electrically that goes to hundreds of cycles

then we have the eeprom that's electrically erasable that goes to hundreds of thousands of cycles flash memory as you may already know goes to millions of times. And finally we have another technology that's called fuse which is also one time programmable in the picture we have a Spartan FBA by Xilinx and Altera cyclon FDA. They are both implemented with static ram. The first shown in the list.

WHO MAKES FPGAS

So who makes a bigger case like I said we are pretty much stuck with two very very good manufacturers. First we have silence at the left and in the picture we have a Virtex 2 FBD which is a very advanced one.

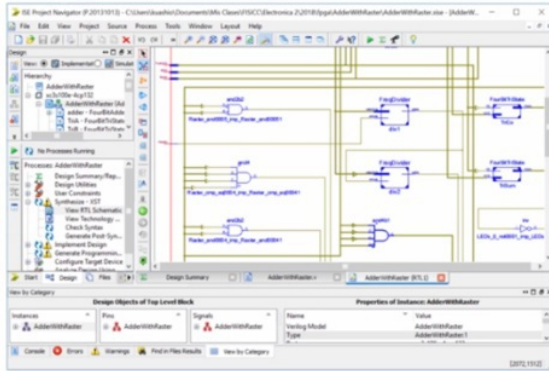
Who makes FPGAs?



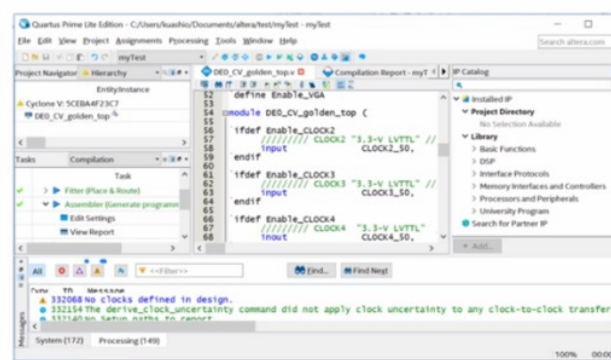
And as a fun fact sellings is said to be the inventor of the FPGA technology. And at the right we have Intel or Outtara as their former manufactured is called Altera was acquired by until recently but they have been the top competitor to sailings for years in the past. Both solutions are very good and very well accepted in the market. Actually these two companies have control of I'm guessing over 80 percent of all FPGA in the market.

Development Tools

Xilinx ISE/Vivado



Altera/Intel Quartus



So there are other manufacturers but the most popular by far are these two as far as development tools go at the left. We have sellings ice or Vivanco which are the development tools. Ice is an older one but still widely used and Vivanco has been around for some years now and it's still development. By selling used to the other right. We have Intel quarters which is the tool originally by Ultegra and now Intel and both tools do the same thing but for different targets deciding stalls workforce silence Figueres. And the Altera tools worked for Altera FPGA and this project will concentrate on Althora tools but their use and the whole toolchain is pretty much the same.

WHAT HARDWARE CAN BE IMPLEMENTED WITH AN FPGA

And if you're still wondering what type of hardware and FBA can implement. Well you can implement Simbel company Shanell logic such as others multiplexers. Here's an illustration of the full adder and forbit adder implemented in this very low project.

o video subtitle tools view help

What type of hardware can an FPGA implement?

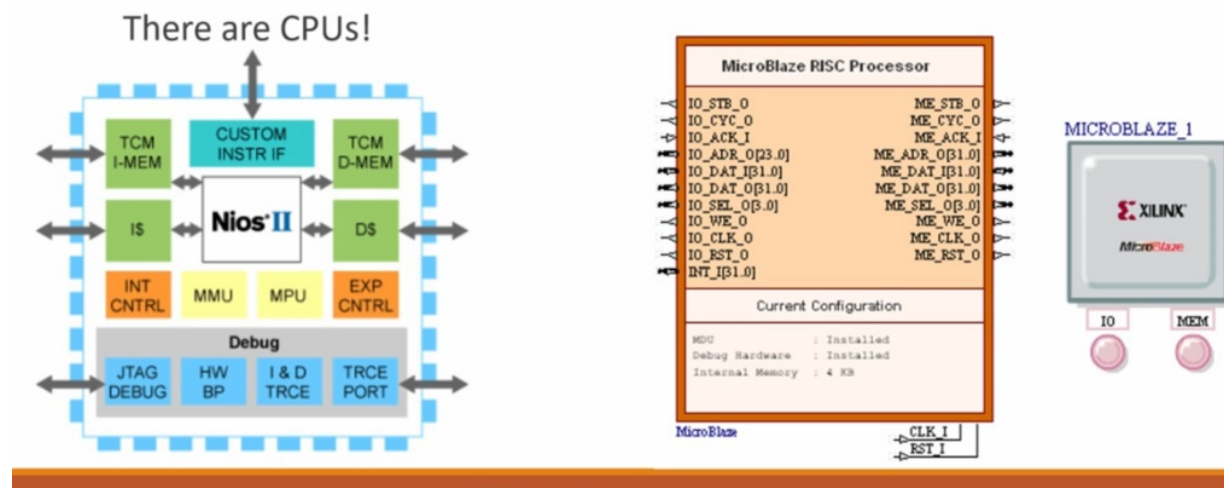
- Simple combinational (adders, muxes).
- Sequential logic.

The image contains three circuit diagrams. The leftmost diagram shows a breadboard implementation of a 4-bit adder using integrated circuits (ICs) and a 7-segment display showing the number 2. The middle diagram is a schematic of a 'Full Adder' built from two half adders and an OR gate. The rightmost diagram is a schematic of a '4-Bit Adder' built from four full adders and half adders, with inputs A3A2A1A0 and B3B2B1B0, and outputs C4C3C2C1C0.

The first one in the series you can totally implement that one on an FPGA and you can also implement sequential logic which is for example the corner in that illustration we have a counter with a display that takes in a 4 bit number and you know what if you have combinational logic and sequential logic you have everything you

need to design any digital system. So what's the limit. Well the good news is that there are C-T use out there and they are not the limit at the left. We have a block diagram of a neo stewe system. That's the most popular soft processor or bioterror. Now Intel and at the right we have an illustration of micro blades which is another very popular soft processor.

What's the limit?



This one is by Xilinx and you may be guessing each of them is suitable for each of their manufacturers FPGA. So I hope you get the idea and are now excited at this point because you can do just about anything you want with an FPGA as long as it's composed of digital devices and additional condition you shouldn't worry about just yet is the size of your designs. The memory inside and if BGA and the number of gates or logical devices inside an FPGA are for the most modest of them equivalent to hundreds of thousands of gates. So you're OK with any device we are going to design in this project but be aware that if you are going to instantiate a lot of microprocessors you may find a limit into what fits inside your SBT. So doesn't all of this sound fun to you.

WHERE TO GET A BOARD

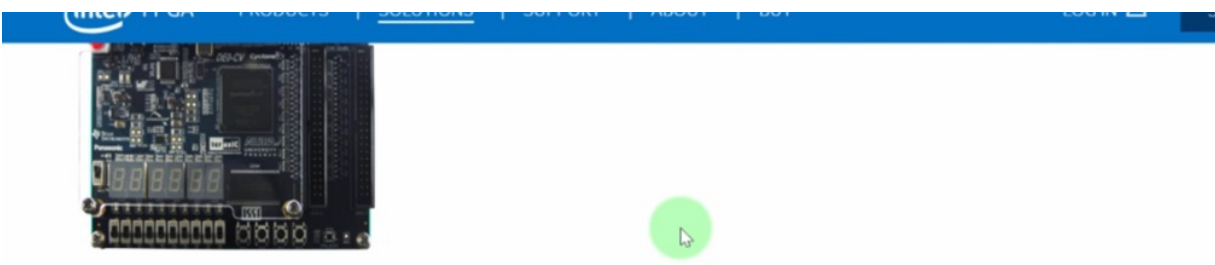
All right. First let me tell you about the board we we're going to use. This is D.L. Tara the 0 dash CV board and after googling it you can see that there are several results. And let me show you a few of them. This is the Web site for Tereszczuk which is the developer the manufacturer of this board and this you can see here it is. And the price as of June 2013 is \$150.



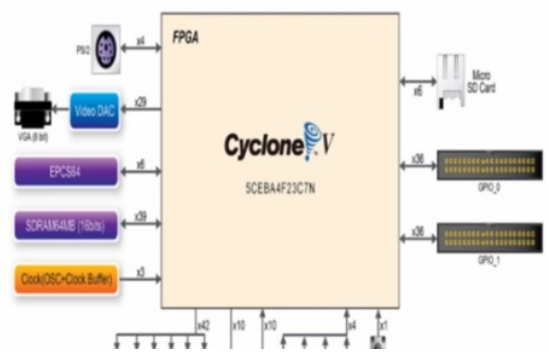
The screenshot shows a web browser window with the URL www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=921. The page features a navigation menu on the left with categories like 'Arria', 'Cyclone', 'MAX', 'SoC Platform', 'Bundle Solution', 'Robotic Kits', and 'All FPGA Main Boards'. The main content area displays a blue DE0-CV board with a green mouse cursor pointing to a component. To the right of the board, there is a price list: '(Currency: USD) Price: \$150' and 'Academic: \$99', with a blue 'Buy it now' button below. A 'Like' button indicates that 13 people like the item.

And if you are a student and you can prove that then you can get it for \$99. So what's in the board. Well it has a set of 10 total switches also 10 Leds right next to the switches and it has 6 7 segment displays. Here is the FBD. And you can see here it has a hard plastic cover so that you don't touch the sensitive electronics in the board.

And it also has four push buttons. It has a bunch of connectors. Here's a piece to connector EVGA connector. This is a slot for the card and it has some other sets of things right here so that you can connect this to other models that are compatible with this board now going to Alturas website as you can see.



Block Diagram



This is Intel branded but it's still the Altera website. Here is the board and you can find all of the information you may want to read about this board. And here the price is a bit different. And finally let's go to some vendor. This is Digi-Key. And here we'll have to check the actual item which is this one so here the price is the retail \$150. So if you want to get one of these boards I would recommend any of these links. And once more if you are a student then you can get a special price for it.

BOARD UNBOXING

So this is the box the Altair or boardgaming. This is from the university program and the art is pretty unique. So the board comes with an AC DC adapter and a U.S. Be A to B cable. So here's the board.



As you can see the hard clear plastic cover stands out and look at this sticker right here. It says download these Ciro C-v CD from this address.



So I guess we're going to have to visit that address to get the software and digital material that is supposed to come with the board because it didn't come with a CD here that this place switches. It's a bit hard to notice the Leds but here they are. Here are the switches and the connectors. Speaking of which this is the S2 connector veejay. No this was B connector is for programming the board. You don't think it's available for your designs.



Here's the DC input. It has a power switch and a very important switch right here. This says run on top and Brugge for a program at the bottom will use this switching a bit o. And here's a micro SD card socket and the general purpose connectors.

THE DE0-CV BOARD WEBSITE

So now let's visit the Web site in the sticker that came with the board that is C D dash the 0 dash CV. Doctress ACT-UP come here this. So once more this is in that resk Web site and the original contents for the CD were these files. So here are three elements in this CD-ROM table. First this one says the zero CV CD-ROM and this is a set of demo applications you can download to your board. This is the only one I recommend that you download.

www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=921&PartNo=4

DE0-CV

Overview Specification Layout Resources Demo Kit Contents Order Now

DE0-CV Board

Like 13 people like this. Be the first of your friends.

Documents

| Title | Version | Size(KB) | Date Added | Download |
|--------------------|---------|----------|------------|----------|
| DE0-CV User Manual | 1.1.2 | 4738 | 2015-05-08 | |

CD-ROM

| Title | Version | Size(KB) | Date Added | Download |
|------------------|---------|----------|------------|----------|
| DE0-CV CD-ROM | 1.2.1 | | 2015-07-03 | |
| ControlPanel | 1.0.1 | | 2015-05-04 | |
| Quartus Download | | | 2015-03-17 | |

Please note that all the source codes are provided "as-is". For further support or modification, please contact [Terasic Support](#) and your request will be transferred to Terasic Design Service. More resources about IP and Dev. Kit are available on [Altera User Forums](#).

Other course resources you might interested:

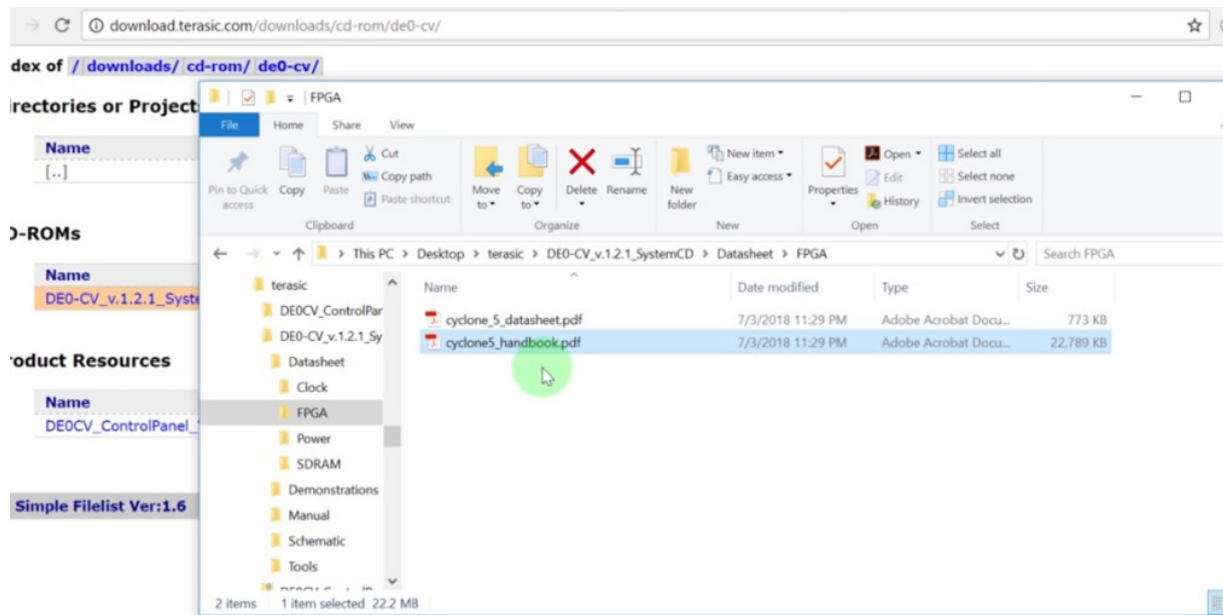
School: Cornell University/Senior Lecturer - Bruno Land

The next one is the control panel. It's our fancy Windows application that has controls for you to turn non-elites or to know the status of the switches in the board online. But really it's just a toy program. It's not a design that you will make. So you can feel free to skip this one.

Besides it requires quarta stew which is a previous version of Quartus and it's not compatible with Gordis prime 17 which is the one I am using and the one we will use in this project. So once more maybe you don't want this one. And here's a download for Quartus. I don't recommend doing this but rather going to the White House Web site. So let's download this CD-ROM file as you can see this sent us to some other page I rather dated Bache it seems and this is the zip file we want. So let's download it.

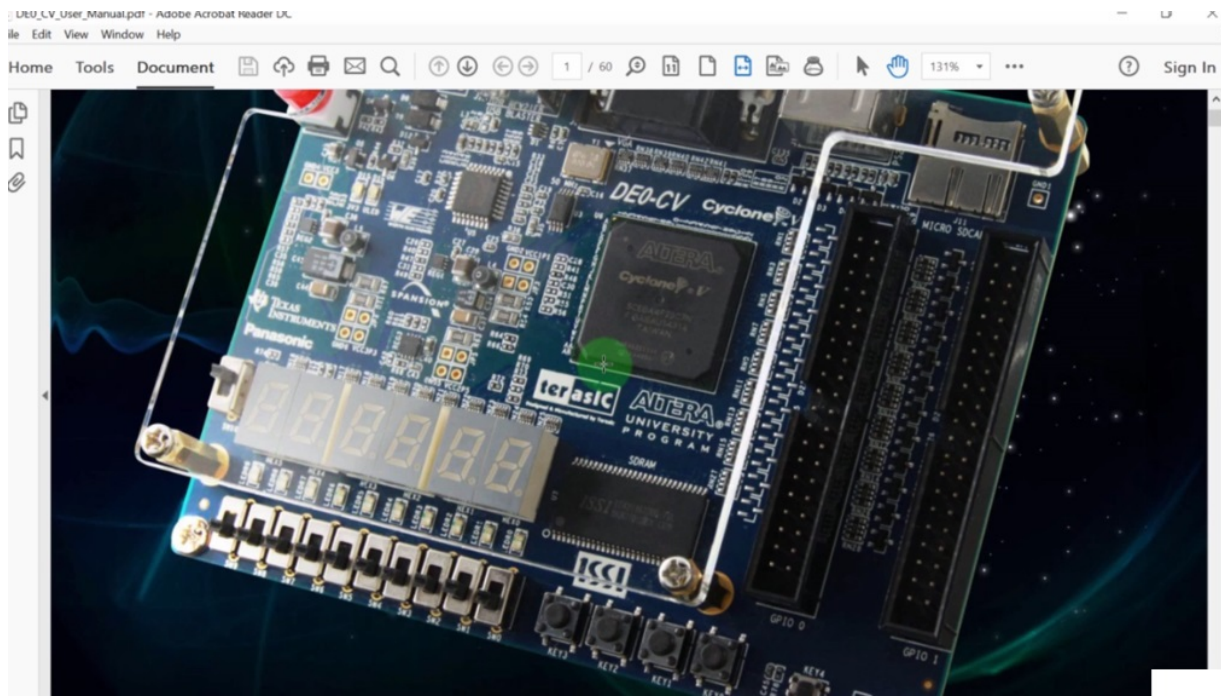
DE0-CV BOARD CD CONTENT

So after extracting all of its files here we have the directory structure. And as you can see we have data sheets for the clock circuitry for the FPGA.



So here's the data sheet for the cycle on five FPGA that it's onboard the data sheet is the brief file with only 48 pages. It summarizes the electrical mechanical timing and temperature respect's of this chip No the handbook is the rather long file with one thousand ninety one pages and it has all of the information you may want to know about this FPGA but by all means feel free to just skim through this file or even skip it because this is very very involved information that you will not even have to know about in most cases. Actually for this

project we are OK with just the information of the board as you will see this information it's good to have handy but that's it. We have some data sheets for the circuitry related to power and we have the SD RAM chip also here detailed in its own data sheet. Here are the demonstrations which you can download into your board. We will see how a bit later but for example here is the default application that comes loaded into the board. If you want to load it back here it is and there are some other cool demos here.

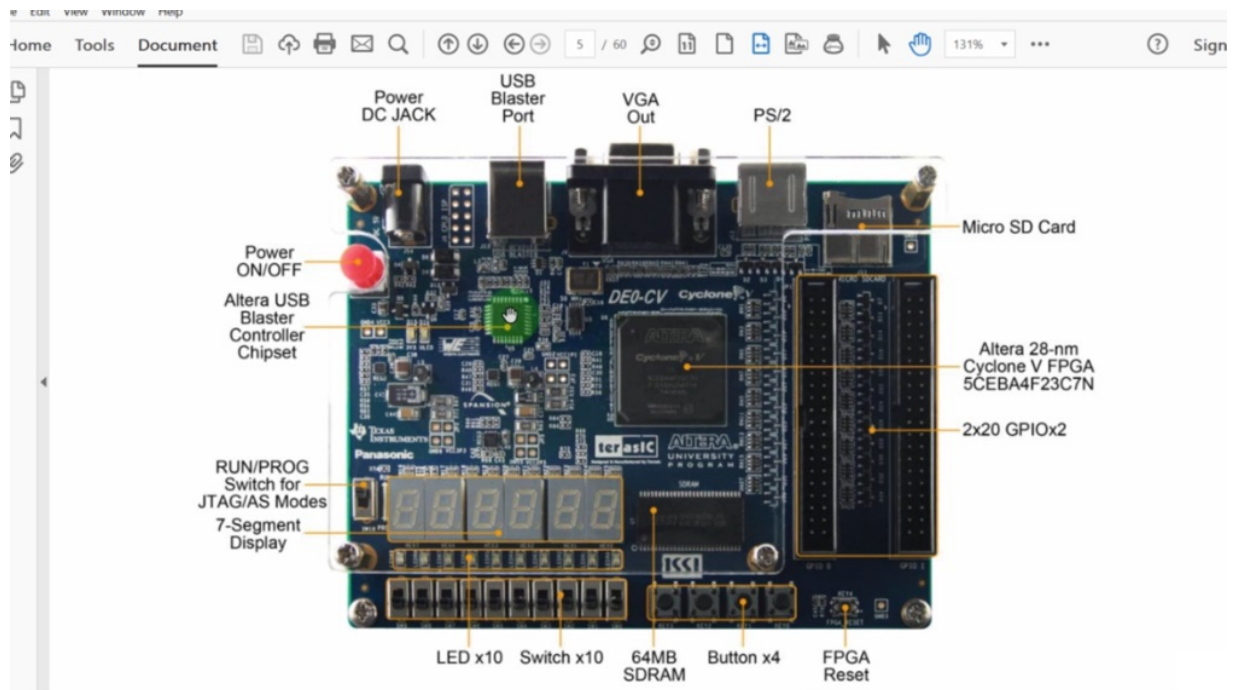


Here's a P.S. to them all. As the as the RAM D-Ga. answer one here's one that uses the Neals microprocessor. By the way here's the manual for the board. This is the file we will skim through. So this is the board manual not the FBI's manual or any of the other integrated circuits onboard. But this is the manual for you for the use of the whole development board. This is much easier to read document because it's for humans. It's only 60 pages long and we'll see it in a bit. Here's the schematic for the board if you care to look at it. So this is only if you want to know and expect a connection and whether or not it's OK to connect something to your board and not break anything. So that's the only reason they see I use or may want

to take a look at this file. And finally we have some tools here. Here is that the control panel application which requires a to do so. We are better off without it. Here's the message. It didn't find Quartus although it is installed in this computer but the newer version is all yes. All right. Let's not use this one. And finally here is another two named system builder. So this is actually a nice application. We will see a bit later. It's the easiest way to create your designs. At least for this board. So that's it. Make sure you download this file so that you can follow along with several examples. I will guide you through.

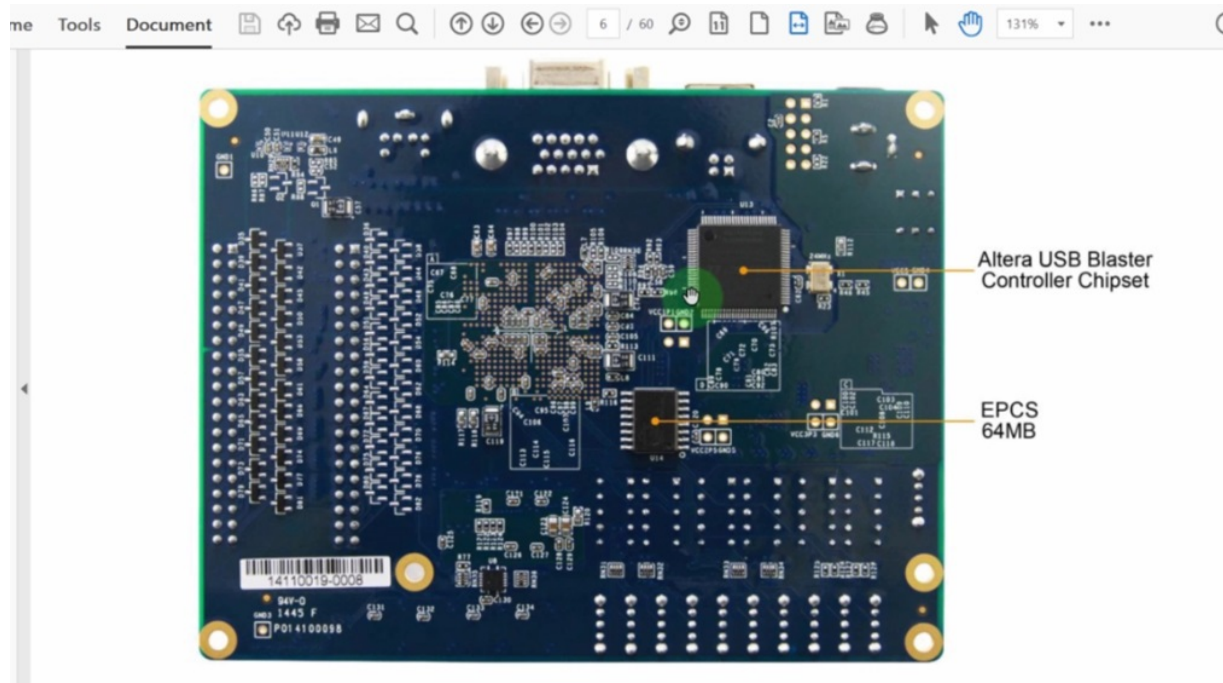
SKIMMING THROUGH THE MANUAL

Now that's browse through the User Manual of this board. This is the best way to get to know any hardware. In my opinion so the first thing I want to do is to show you all of the hardware that is on this board formerly. So here's a nice illustration as you can see this board has a lot of connectors. These connectors are general purpose. This is the FPGA and this very special chip right here is the U.S. bead blaster.



So this chip works with this U.S. connector to program the FPGA in a way that you'll learn shortly. Here's the power input for DC. This is 5 volts and a power switch. This switch right here the one that reads run Brugge is for running the application that you download through

the U.S. Blaster into the FPGA or for programming some applications that will permanently stay in the FPGA board. I'll tell you how later. Here are the seven segment displays. Here are the Pleiades the switches and the push buttons. This is Diest DRAM chip. It has 64 megabytes and we have an FPGA reset button at the bottom and the peripherals you are free to use are the VDARE output the S2 connector and the micro SD card adapter at the bottom of the board. Here we have the USD Blaster controller chipset. This is a very specific circuitry required for the programming of the board and this chip right here is a nonvolatile memory that can hold the whole configuration the whole circuitry you want to send to the FPGA but this is a nonvolatile memory. This FPGA does cycle on five stores the files you download to it in static ram. That means that when it loses power it loses the circuitry you sent to it. The reason they are implemented with SRAM is because it is fast and the way you download an application to stay permanently here is into this chip right here. So for the most part you may want to download your applications directly into the FPGA so that you can make changes and work rapidly. And when you are sure you want your application to stay on the board that's when you send it to this chip right here. All right let's move on to the next thing I want you to pay attention to. Here's a nice pictographic block diagram that shows us all the elements in the board. Now if we scroll down to chapter three this is where we can learn how to configure and how to download applications into your board. So here it is you have two options. One is the. Tag programming and the other is the active serial programming. These are the two ways I told you about JTA is programming the FPGA directly and active cereal is programming the nonvolatile memory. So the first method and the one we will primarily use in discourse is to program from Quartus into the U.S. be Blaster directly to the Altair a cyclone the FPGA that happens when you have the run programming switch in the run position. So it is this switch that has to be in the top position.



The one that says Run and whenever you want to download your application permanently into the board you have to put this switch in the bottom position to program position and so the output will go into the nonvolatile memory. It's not just a nonvolatile memory it's a serial configuration device that's the name of the chip. And what it does is that when the board powers up it downloads all of its content into the FBA every time you power up the board. So it's part of the boot up procedure of the development board. Oh and another technical difference is that to program the FPGA you will need an S or with file a file with the S or with extension where s to program the serial configuration device. You will need a file with the B or with extension. So that's it for now. We will come back to this user manual when we need to learn about how some elements of the hardware are connected and what a logical one and a logical zero does to your application for specific elements like Ltd's or switches.

THE FPGA DEVELOPMENT PROCESS

Know that we have briefly covered what FPGA you are and a bit vaguely how to program them it's time to learn the gory details of the FPGA development process.

In this lesson: How to Program FPGAs

- The Steps you Need to Take
- Where the Compiler Takes On
- How to Download your Code into the FPGA

In this project we'll learn how to program FPGA and I'll show this to you in three steps. First I'll tell you about the steps you need to take. Then the steps the compiler takes and finally how to download your code into your SBA. So let's get started.

THE STEPS YOU NEED TO TAKE

So here we have a summary of the steps you need to take in order to make an FPGA digital design. First you create your project. Indeed you have decided to use then you write your code in very low or VHDL. Next you assign the pins that is assigning where your input and output ports for your Top Model are going to be connected to in the FPGA. Next you specified timing constraints.

The Steps You Need to Take



Now this is one of the most important steps in high speed systems and even in systems with modest timing requirements. Next you hit compile and I'll tell you all about what happens here. Next you don't know your design into the FPGA and you usually do this by hand. So let's take a deeper look into these steps.

CREATE A PROJECT

So the first step is usually to create a project in the idea you have decided to use. Chances are you will work with projects and projects are a structure of files and they usually work within one folder in your computer where you have a project file and that project file is usually a text file. Maybe SML or Jason that contains all of the information necessary to use the rest of the files to produce your applications. It has all the settings you set in some dialog box and all the decisions you make. Well all of this information goes into the project file. And here's a list with some of the information you have to provide to the IDE so that it knows what you want your project to do. One of these details is the purpose of your project.

Create a Project

- IDEs work with projects.
- Information you provide:
 - Purpose (Simulation or FPGA)
 - Target FPGA
 - Target Board (If supported)
 - Primary Source Language/Method
 - Source and Libraries (IP Cores)

Typically you may want to simulate your project or download it into an FPGA which is called implementing the design. If you are going to

implement it you have to specify which FPGA you want your design to go into because I see no vendors provide several different FPGA of their own. The target board is sometimes specified especially if the vendor provided a project file or more information on it. So you may work with VHDL or very long you are sometimes asked which language you prefer so that the new files and the libraries you include may come in that language is just for convenience most IDs are multi-lingual in that they support VHDL or very low modules alike. You can also specify which IP project you want to support. Remember IP project are readily available modules pretty much like software libraries all of them require a license. Some are paid and some are for free.

WRITE YOUR CODE

The next step is to write your code. Like I said most ideas support very long and VHDL but it's very common that you can enter your schematic diagrams either and RTL schematic or a technology schematic. So the RTL schematic is the logic diagram with Gates and whatever you want your system to behave like. Whereas the technology schematic it has the actual parts that are inside the FPGA.

Write your Code

- Most IDEs support:
 - Verilog
 - VHDL
 - Schematics (RTL and Technology)
- You have to include all modules in your project.
- Lots of IP-Cores available.

So if the FPGA does not have let's say and gate Well they won't be available in your technology schematic but they may be shown in your RTL schematic. Basically what this means is that RTL is true to your design. Whereas technology is the best fit that the software came up with for your design. When you write your code it's important to include all the modules you are going to use including

I.P project and there are tons of those available. And I recommend that you look into your ID so that you can see which you are free to use right away. A bit later when they show you around Quartus I'll show you where you can get your IP project.

ASSIGN PINS

The next step is to assign parents so further than just a signing where each signal from your reports is supposed to be connected to or which bins are supposed to implement your sports. There are several other details you need to specify when you assign the pins. Here's a short list which is by no means complete well. This first item is locating your pins. This is the obvious reason why you need to assign the pins. But there is a long list of so-called attributes. These attributes are several electrical details of the logic you want to implement on your pins. For example the wind direction if you want it to be an input or an output or a bidirectional been an input could be a schmitt trigger input that's an input that has just the Rices meaning several things but these inputs are used usually to implement oscillators or to interface with analog devices. You can specify if you want your inputs to have pulled resistors so that if you leave the nodes floating you don't connect anything to an input. It will have a default state as a logical one where you would use a pull up resistor or if you want a logical zero as a default then you would use or pull down resistor the slew rate is the rate of change on an output. And this has several implications.

Assign Pins

- Some important information about pins is necessary:
 - Pin Location
 - Pin Direction
 - Schmitt Trigger
 - Pull Resistors
 - Slew Rate
 - Many, many more!

- This info resides ultimately in some text files.

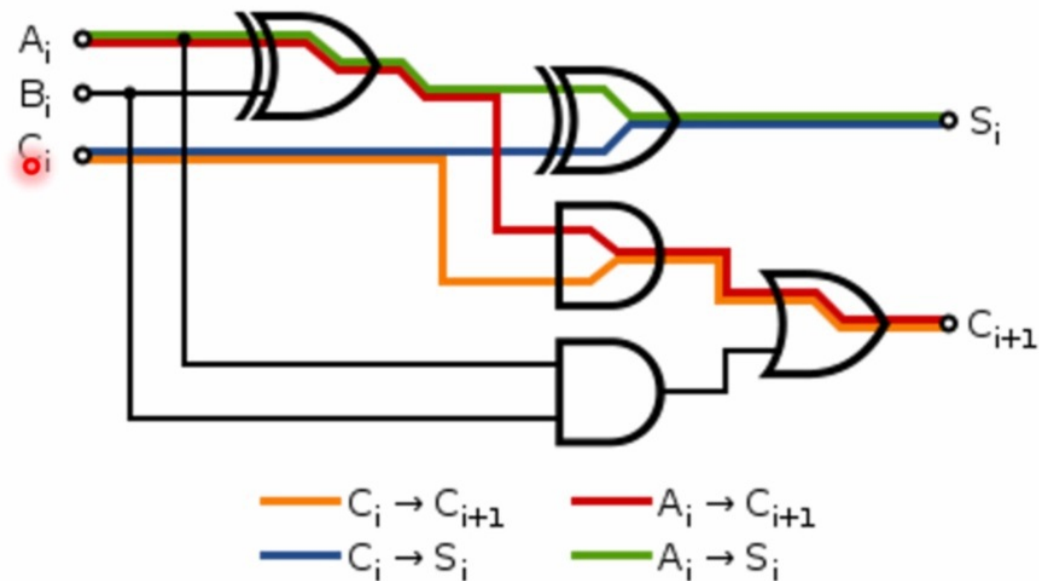
Normally the impedance of the surrounding hardware determines if you need a slew rate control or not. And there are many many many more input output attributes and I have somewhat good news and that is if you are going to use our development board the vendor will probably already have set up a project for you. And so the assignment is usually already done and you have access to some file that does this location and configuration of those pins. This is just something that usually happens when working with development boards. Of course if you are working with your own board that you designed you would have to specify all of these things because you have designed all that hardware. Then again you wouldn't be taking this project if you were at that point and once more this information is ultimately saved in some text file this text file may be the actual project file but it's usually enough file that is openly known to contain all of those pin assignments. So as you can see this can be tiresome because you have to specify a lot of things for a lot of pins. And so this is usually done in a separate application. Remember a Navy is a whole suite of different programs that perform different operations for a common purpose and that is making FPGA embedded applications. And so there's usually a separate program that belongs to the development suite that does all of this and this

program is included because well on the one hand it's easier for you when you have a graphical user interface. And second your printed circuit board may benefit from having certain locations for certain signals. And so it's often very convenient to have a graphical view of the chip package where you can see the pins and where they are located so that you can make a good estimation of how your printed circuit board will end up looking like.

And we can go through all of these bins and they invite you to do that and try to read through the legend and try and understand what all of these symbols and assignments mean. So this is the package of the FPGA as you can see this FPGA comes in a very complex package that has pins all over. This is a BGA package or ball grid array. And these are all the signals that are available to use in your code. As you can see their names are G.P.O. Hex as in the hex displays LCDR and seeing the Ltd's and so on and so forth. Well this is where you could modify your PIN assignments if you want for this project we will use generated projects. And so this won't be necessary but it's nice to come take a look at this so that you know exactly where everything is located.

SPECIFY TIMING CONSTRAINTS

The next step is to specify timing constraints. So this is one of the most important parts of the process. I can't say this enough times and for me to explain to you why this is important in FBA D.S. especially we have to go back to the sad reality of digital systems and that is propagation delays in this illustration. We have a diagram showing some logic but the bath's that these input signals have to go through to get two of these output lines are shown in different colors. By the way this is a full aether as you can see but once again in four different colors we can see four different stories. The signal he goes through this X or iGate then it goes through this and gate and then through this or iGate or a big deal but that time it takes for this signal to go from its inputs and have an effect on this output is called the propagation delay of this gate.



Next we have the propagation delay of this gate. When this result in signal enters this gate and causes whatever it's supposed to cause in this output and in the same way we have that this signal will have some effect on this output and this will happen after a certain time and we don't have any guarantees that all of these gates have the same propagation delay but they may have similar propagation delays. However some Gates may have different propagation delays especially surrogate's because as you may remember the XOR operation is a compound logic operation. So this gate and this gate may have each a propagation delay that is higher than the propagation really off and gate or and or gate. So what does this mean. This means that the time it takes for me to have an effect on the carry output or the carry impact of the next stage. Right here all the time it would take to have that effect would be the preparation delay of the X or plus the preparation delay of the end plus the preparation of the other and but that's not all. What if we want to know how long it would take for a survive to have an effect on this output. Well that would be the preparation delay of the first X or gate Plus the preparation delay of the second surrogate that's the green path. This one shown right here. So I hope you can see that these two times are different. The red path takes x4 glass and glass or

propagation delays whereas the green line takes X or propagation delay plus X or propagation delay. This means that the yes output and the C output won't change necessarily at the same time even when you make only one change in a because the different gates may be different times to respond and it gets a bit worse when you consider C as you can see the paths are only cause for a and see right here B has no color in it but it's just not shown. So if we take a look at c c has to go through only the XOR gate to have an effect on s that carry input only has to traverse this gate to have an effect on this output. And it also has to traverse this and gate and then this or a gate to have an effect on the next Gary line. So the carry signal the carry input signal has visibly a shorter time or a shorter overall propagation delay in this system to have an effect on the S output and DC output where S A has to take longer because it has this X or to go through first that means that when you change a b and c even if you make the best effort to change them at the same time these two signals don't have any guarantees that will have changes at the same time or even that these changes will be stable. Let me show you what I mean.

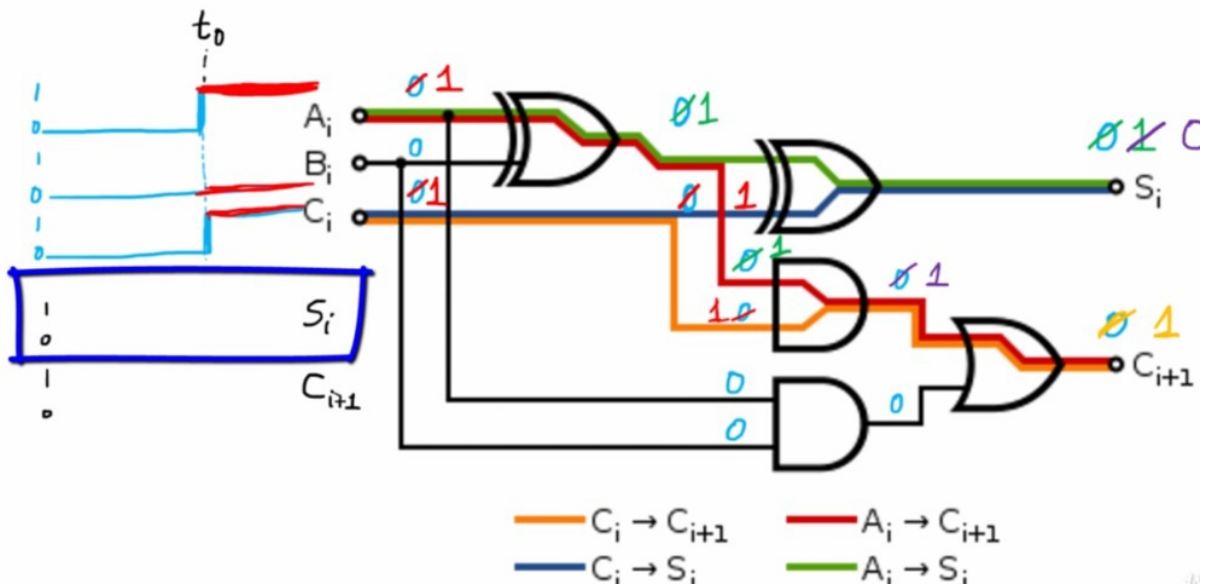
PROPAGATION DELAYS

EXAMPLE

So let's say a was zero. And at some time we changed it to one let's say B was always 0 didn't change and let's say that C also was 0 and changed to 1. At some point and please bear with me let's say that this racing incident was the same for both gates. So please excuse my drolling and this time I'll call it the zero. OK. So let me show you what happens after you zero in red. So in the diagram paints and red. So take a look at this. If we had 0 0 0 entering this system then the signals A B and C would look like this to zeros in this X or produce a zero this zero in the C input goes as an input to the gate producing another zero in some on the other hand. The two inputs of the lower end gate are zeros and the upper and gate also has zeros. So the output 0 and the OR gate outputs are zero. No biggie here. Now let's consider what happens right after 0. So I will show what change here in red so first we changed a 2:1 and see to one that's the first thing we changed. And by the first thing I mean that the change in C is also visible right here at the right. So this zero that's entering the lower input of the X or date is now a 1. So this XOR gate is as of this moment as of 0 now experiencing a 0 in one of its inputs because this XOR hasn't responded to this one. This is still outputting this 0 but the one we just changed here because remember we said that these changes happened at the same time. And so C has an advantage over a. In this second X or gate. Meaning that when this x4 sees a 0 and a 1 it will have to put now a 1. But I don't want to change this right now because I am only showing you what happens at the zero. So let me mark this in some other color to show you that very briefly this signal will be the result from this zero. And this one and which will be that result will zero in and one in and or produce one. I will use green for this. So we have a 1. So this signal used to be a 0 then became 1. Now let's see what

else happened when we changed. See to 1 we'll see comes as an input to this and gate the upper end gate. And so we get a logical one here. So still at this moment nothing changes for this gate because it's an end. So it's still outputting it's 0 and then right after the propagation delay of the first XOR gate we get something new and you know what. I'll use green. And there's a reason for it. Will this change from 0 to 1. Kurt when this gate noticed having a 0 in one of it's input. Any one on the other. So the change from 0 to 1 in its lower input is causing a change from 0 to 1 in its output. But when this one finally is changed is when this gate has responded. So the propagation delay of this extra gate has passed and will this other X or gate is supposed to have the same propagation delay. And so I will show this in green this change.

Propagation Delays



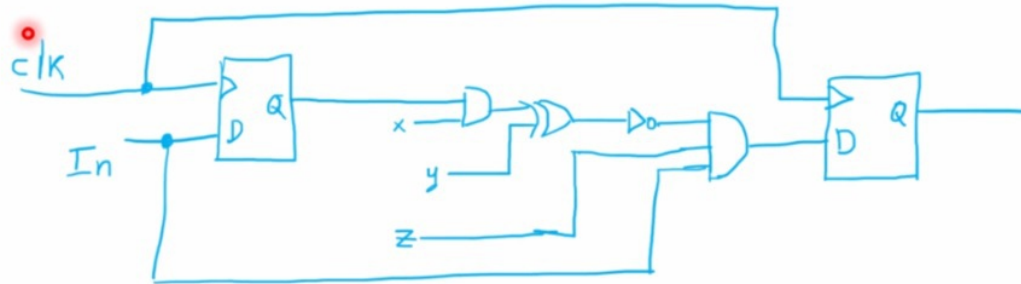
So this signal will change at the same time as this one. So this one goes to one. And now this X or is looking at a logical one here and the logical one here. So it's now again forced to change this into a zero. I will show this in some other color. But let's see what happens with this one. Also this one comes to this input of D and Kate and so we will change this signal to one race one. And so after some time

this signal will change. This will be a bit earlier than when this signal changes. And then this change will cause this we wondered will appear here will cause a one to appear here because this is an --. So let me use purple now. The output of the X or gate will now be the 0. It has to be the output of the upper and will be the one it has to be. And after some time the output of the Gary line will be a logical one a logical one. So the real problem right here is what happened to this output as you can see this output. Didn't get just one change even though we changed these three signals at the same time. Or rather wait and see changed at the same time. We didn't get one solid change but a set of changes and over time these changes won't look synchronous with each other. So this may be very well what happened with these two signals. I'm not saying it is exactly because I'd need to know the propagation delays. So let me show you right below this diagram what these output signals may have looked like. So this signal was originally 0 and right after this change in 0 2 it went from 0 to 1 and then back to zero. So let's say that happened like this. So here's the one it's momentary and then it stayed at zero. Let's say it took this long. No for the see I plus 1 signal. The story's different. So let's say this signal changed a bit later than the sum signal so once more. Please excuse my drawing but what I mean to say is that this signal had some instabilities before it reached its stable state and this signal changed at some different time. Then this signal. And so we don't have any guarantees that a combinational system will behave in a synchronous manner.

TIMING IN SEQUENTIAL SYSTEMS

And what's so wrong about propagation delays. Well the thing is that sequential systems do require those times to be considered. So let me show you a small system so that you can see what's really going on and how we need to take care of these delays and be aware of them. Let's say we have a clock line here entering EDI Flip-Flop and let's say that the signal this The Flip-Flop output goes to some logic but more importantly let's say that this signal goes into another deep flip flop as its data input. And we have the same clock signal going into it. As you can see we have some inputs here. X Y. And why not Z which go to this final and get this in signal also affects that and gate. Do you see where I'm going with this. Well the problem here is that no matter what the logic in x y and z is this input has a huge advantage over all of these other signals because it goes from the actual input into this and gate which actually has the same advantage as Z but why has to traverse this X or gate. Then the inverter and X has it even worse it has to traverse the and gate then the X or then the inverter and so is the situation of this output. So if we want this and output to be valid when we receive a clock edge in this flipflop we need to allow the propagation delays of these gates so that these signals have an effect on the end output and we need this signal to be stable.

Timing in Sequential Systems



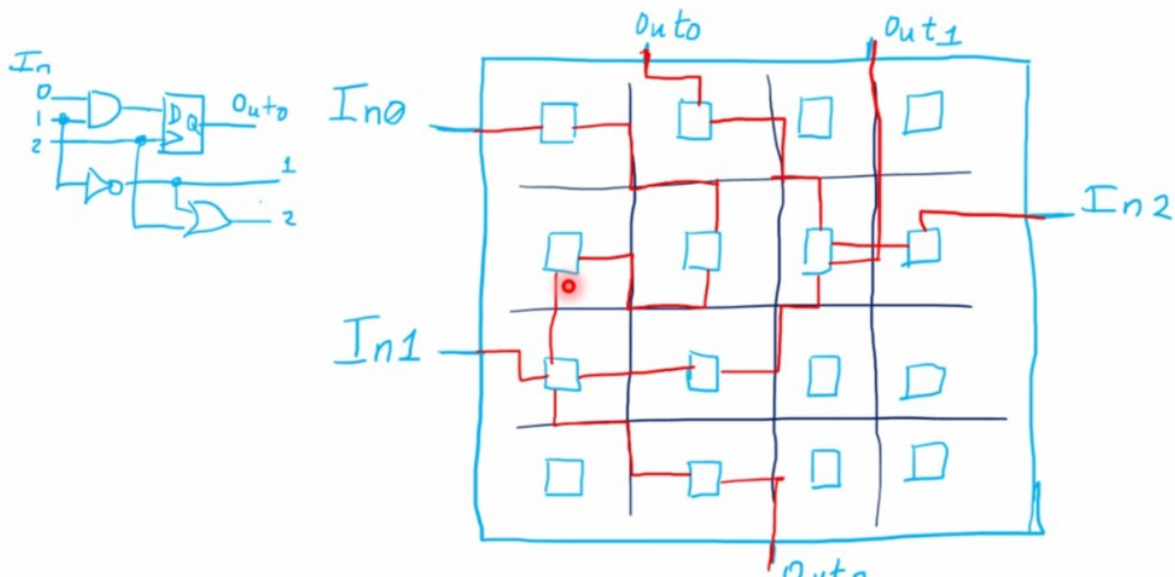
That is not to change because of some delayed state has cost changes and finally reaches the signal we are interested in. So the project here is that for this Flip-Flop to work properly in this system we may naively think response with no delays. Well we have to take those delays into account and we are limited in the clock frequency. This can only work at some maximum Gluck's frequency because if we try to do it faster this final Flip-Flop will not wait for the signals that are still going through the gates. And will this would malfunction. So that's why we have to specify the timing constraints. We have to let our software know we have to inform our software about these delays and tell our software hey this clock line I expect this line to be at I don't know 10 may hurt. So you better create logic right here. That will comply with that deadline. So the deadline would be the period of the frequency you want in your clock signal. And as you may imagine one of the worst things here is what they've shown to you right here. These unstable glitches could be a very bad thing if we are using frequencies higher than the ones our system supports because maybe our system because of working too fast may have caught the logical one that was just a momentary effect of the propagation delays right here. Something known as a race condition as all of these signals are in a race between each other to reach the

output port. But you have to let the slowest signal to reach the output. That's why we need the timing constraint. But wait there's more.

WHY THIS MATTERS IN FPGAS

You know why this matters in FPGA. Well let me make a diagram for you so that you can see why an FPGA may be at a great disadvantage in this regards. So let's say we have enough PDA with a lot of blocks. So let's say that we have it blocks here. So mother is here a mother here. So if BGH may have thousands and tens of thousands of blocks which contain hundreds of cells each and that's how we reach the equivalent of hundreds of thousands or even millions of gates. Please bear with my drawing and so let's say that we have some inputs here. I'll name it in zero some other inputs here. I'll name it in one and some other inputs here in two. Now let's say we have an output here out zero. Another outputs here out one and right at the bottom we have output two. Now for this system let's say that I am interested in some very simple logic but that logic requires some gates and some flip flops.

Why this Matters in FPGAs



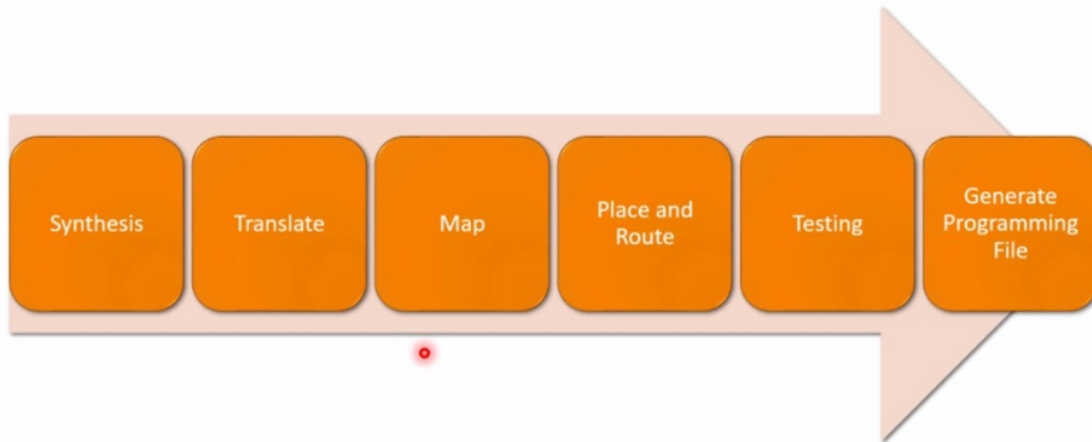
So let me do a little diagram here at the left where we have the three inputs. I don't know I have a gate CA have flip. This is one output. We have another Gaetz here. But here's the real problem. The T that this and gate couldn't be located in any other place and the place will show you here. And there was only room for this inverter at some other point and there was only room for this or and for this the Flip-Flop at some other points here but very inconvenient ways. So let me show you. But say that for the interconnections we have several lines between all of these blocks and I will show the connections in red. So let's say that this input had to go to some element in this block and then it had to go through this line to this part right here to this block in order to now go to this block and then go to this block and then go to this block to finally reach out to then input to go to this block which goes to this block which goes to I don't know this block reaching out at zero. Now input one goes to this block then to this one then up to this one then right here to put one. So once more what I'm telling you is that maybe this was the poor choice of routing that the compiler came up with and all of these Bath's that were selected and all of these connections will take some time in propagation delays and so we would end up introducing circuitry that doesn't just logically resemble this but it also adds propagation

delays and propagation. The least that are not guaranteed to be exactly similar to the gates preparation's delays we have here unless we inform our software about our timing constraints.

WHERE THE COMPILER TAKES ON

So the next step is to compile. And now I will tell you what the compiler does. So as you can see the point at which we left our design is where the compiler takes on and it goes through a series of processes to finally generate what you want which is a programming file these steps have several names for several vendors but the names. Xilinx uses synthesis translate map place and route testing and finally generating a programming file. The first step is synthesis and this can be simply put as converting your RTL design into a netlist representation that is converting whatever you described in your hardware description language sourcecode into whatever elements are actually implemented in the field you want to use. So for example if you are using logic cells that have lookup tables and you don't have any gates actual gates available then the synthesis tool will turn your design into netlist that you have only lookup tables. They don't have those gates. So that would be converting from Gates to using for example lookup tables. The flip flops.

Where the Compiler Takes On



That's a full adder which are the elements available in the hypothetical logic cells I showed you earlier and as you can see this is one step closer to the implementation and by the way these steps that are called translate map and finally place and route these three steps are known as the implementation of the design. So these are translate map Blaser route and different vendors called these steps differently and even their meaning has changed over time. So what translate usually does is taking several netlist that the synthesizer has output and put them all together into just one big netlist next map selects which blocks to use in order to make it convenient to not end up with a maze of connections. And actually the routing of those connections takes place in here in place and route. So the combination of these steps is finally to implement the design to take all of the net lists that the synthesizer has produced and to calculate where to get all those elements in what locations of the actual FBD. And finally calculates the route it has to make with the interconnections once more. This is way more complicated than the simpler task of a traditional compiler which only has to produce assembly code equivalent to the higher level source code you are using from our C or C++.

TIMING ANALYSIS

The final automated step is the timing analysis. This is a step I called testing right here. And the thing is that the implementation was performed by taking the timing constraints you defined as rules. And it was trying its best to comply with them so the generated circuitry is later analyzed by the compiler toolchain for compliance with these constraints. This is done by a separate program which is the timing analyzer in Quartus.

Timing Analysis

- The implementation steps take timing constraints as rules
- The generated circuitry is analyzed for compliance with constraints

This one is called Time quest which finally produces a report which tells you if those timing constraints were met. So it tells you if your design is feasible if it's possible to build it. So this part is very important and it's the support the ID provides to you so that you know that your design will work or if it will fail at the frequencies you

want it to work you can always follow the recommendations of that report and even relax your requirements.

PROGRAMMING FILES

The final step of the automated part is to generate a programming file. Intel or Altera Coltart assembly. So the part that tests assemble programming file is the same as generating it once more. This is the binary file that you download into the FPGA. No this file can go into the FBI or it can go into an on board flash memory to download on each power up so downloading your design can be done into the FPGA. This is usually in RAM memory. That's the one included in the FPGA to implement the designs. And so as you know this is volatile. So if we turn nasty FPGA it will lose this application. So this is convenient for development because this is done very fast so you can program your FPGA in seconds and you can try your designs and then go back and make changes and then try again. That's why this scheme is useful. But on the other hand you can download your design into a nonvolatile memory that is usually available in any FPGA system.

Downloading your Design

INTO THE FPGA

- Usually in RAM.
- This is Volatile!
- Convenient for development.

INTO A NON-VOLATILE MEMORY

- Downloaded at power-on.
- This is part of the boot-up procedure.
- Convenient for deployment.

The content of this nonvolatile memory will be download at power on into the FBI. Now this seems as a quick fix as a bad solution but really this is part of the broader procedure. It's industry standard. Everyone does it and that's because the RAM memory of the FPGA is what MAKES IT field programmable. It's the most important advantage of the FPGA. So this method is convenient for deployment of your final application when you are ready to download your application into your FPGA and not change it frequently. Or do you want it to survive power on cycles. Well you can download it into the nonvolatile memory.

INSTALLING QUARTUS PRIME

So if you haven't already installed Quartus Let me show you what you need. So let's look for it. Uncle the one we need is called quarter sprang. So again this is Intel branded and it's the one that says download Intel quarta prime design software.

Three Intel® Quartus® Prime Editions to Meet Your System Design Requirements

| Pro Edition | Standard Edition | Lite Edition |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
| Paid license required | Paid license required | FREE, no license required |
| The Intel® Quartus® Prime Pro Edition software supports the advanced features in Intel's next-generation FPGAs and SoCs with the Intel® Stratix® 10, Intel® Arria® 10, and Intel® Cyclone® 10 GX device families. | The Intel Quartus Prime Standard Edition software includes extensive support for earlier device families in addition to the Intel Cyclone 10 LP device family. | The Intel Quartus Prime Lite Edition software supports Intel's low-cost FPGA device families. |
| Download Now ▶ | Download Now ▶ | Download Now ▶ |

Compare Pro, Standard, and Lite Editions v18.0
For the complete device support list, please visit the [Download Center](#) page.

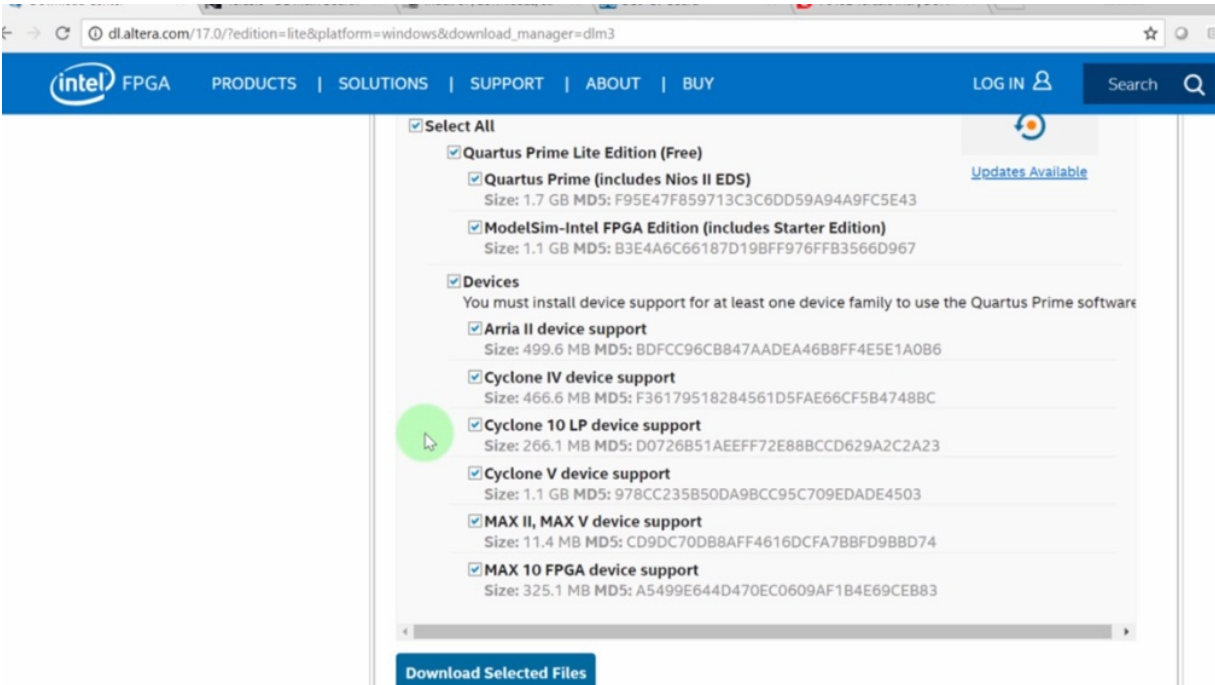
| | Intel Quartus Prime Design Software v18.0 | | Pro Edition (\$) | Standard Edition (\$) | Lite Edition (Free) |
|----------------|-------------------------------------------|--|------------------|-----------------------|---------------------|
| Stratix series | IV, V | | | ✓ | |
| | 10 | | ✓ | | |
| | II | | | | ✓ ⁽¹⁾ |

This is version 18 and as you can see you can download it by going to the light edition right here. This download is free of cost and the others do come with a paid license and so you'll need this one. In this page you can select several releases and versions of Quartus the version I am using is seventeen point zero. You can download it here or if you want. You can download the newest one. I will stick to seventeen point zero. It reloads. And we won the light edition

because this is the one free of cost. You can download it for Windows or Linux. Sorry no Mac.

The screenshot shows the Intel Quartus Prime Lite Edition download page. The URL is dl.altera.com/17.0/?edition=lite&platform=windows&download_manager=dim3. The page features a navigation bar with links for FPGA, PRODUCTS, SOLUTIONS, SUPPORT, ABOUT, and BUY, along with a search bar and a LOG IN button. The main heading is "Download Center" with the subtext "Get the complete suite of Intel® design tools". A sidebar on the left lists various software categories: Design Software, Embedded Software, Archives, Licensing, Programming Software, Drivers, Board System Design, Board Layout and Test, and Legacy Software. The main content area is titled "Quartus Prime Lite Edition" and includes the release date (May, 2017) and the latest release (v18.0). It offers options to select the edition (Lite) and the release (17.0). The operating system is set to Windows, with Linux also available. The download method is set to Akamai DLM3 Download Manager, with Direct Download as an alternative. A warning message states: "You may be exposed to a vulnerability issue if you have installed or plan to install Quartus Prime/Quartus II software from v11.0 to v18.0 to a location with space(s) in the path. See this [KDB solution](#) for more details." Below this, a green checkmark indicates that the software version 17.0 supports the following device families: Arria II, Cyclone 10 LP, Cyclone IV, Cyclone V, MAX II, MAX V, and MAX 10 FPGA. At the bottom, there are tabs for "Combined Files", "Individual Files", "Additional Software", and "Updates".

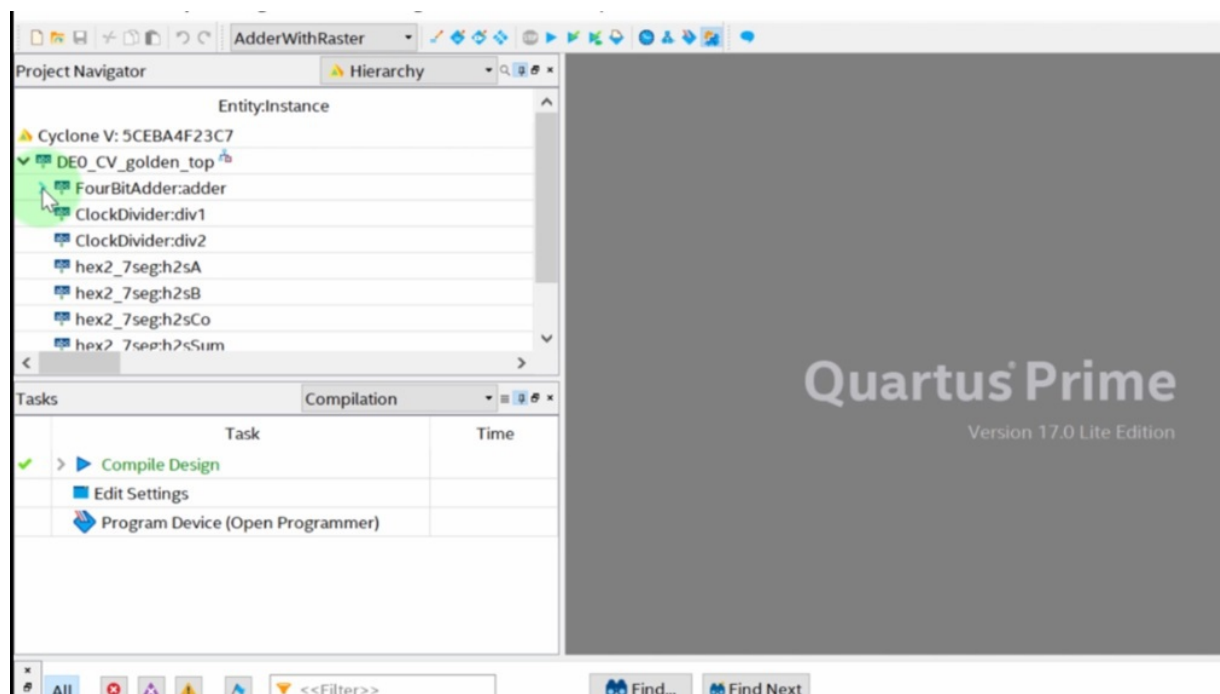
And in this part you get to choose what you want to design for and so you want this quarta sprained. LATE EDITION the free one. This is something you're definitely one you want model sim incase you want to simulate your designs. I recommend that you do. Now for devices there are some devices you just don't want here. If you are only going to follow along with the project you only want support for the site clone the device that cycle on 5.



The rest you can uncheck so you can do this. So these are hundreds of megabytes each and so by only selecting the site. Do I support your saving some space and then you can download those selected files. That's it. The site may require you to log in. So you may want to create a user in the Intel slash Altera website but that's all the support you'll need.

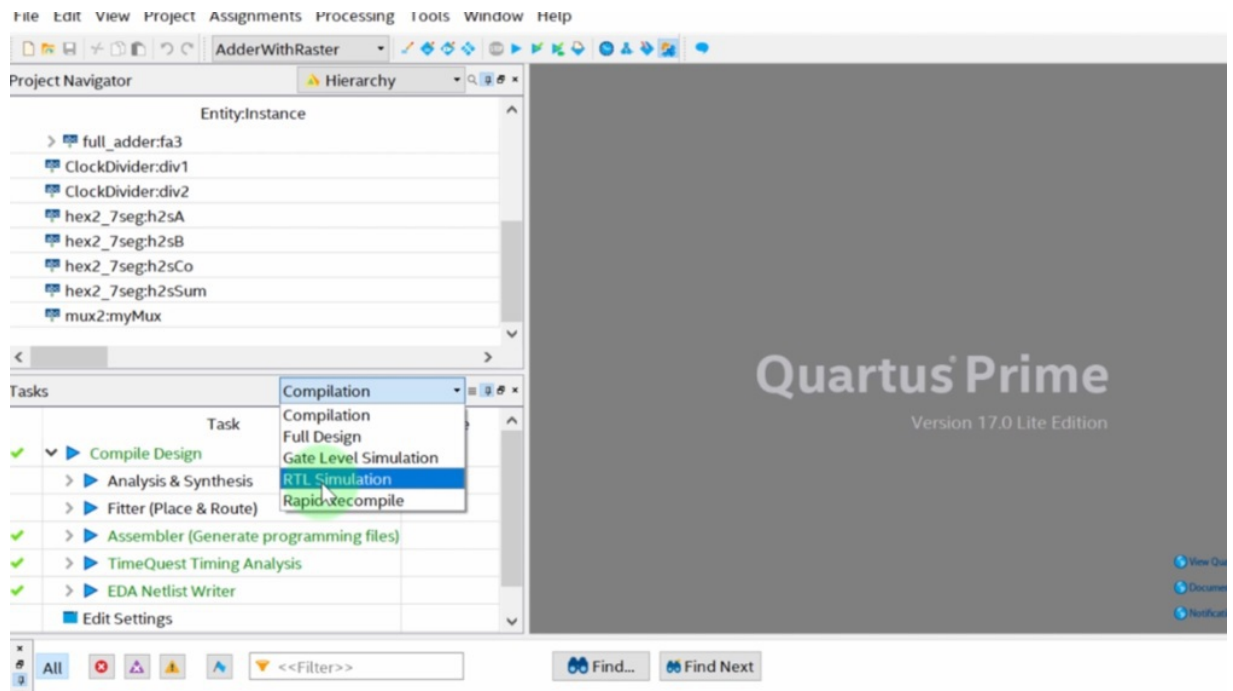
SHOWING YOU AROUND QUARTUS PRIME

And let me show you around. So this is an I.D. and as such it has several sections and several separate applications. I will show you a few right now here I have a project open which I will show you later but for now let me just show you around where everything is. So as in most Adie's at the left we have our design.

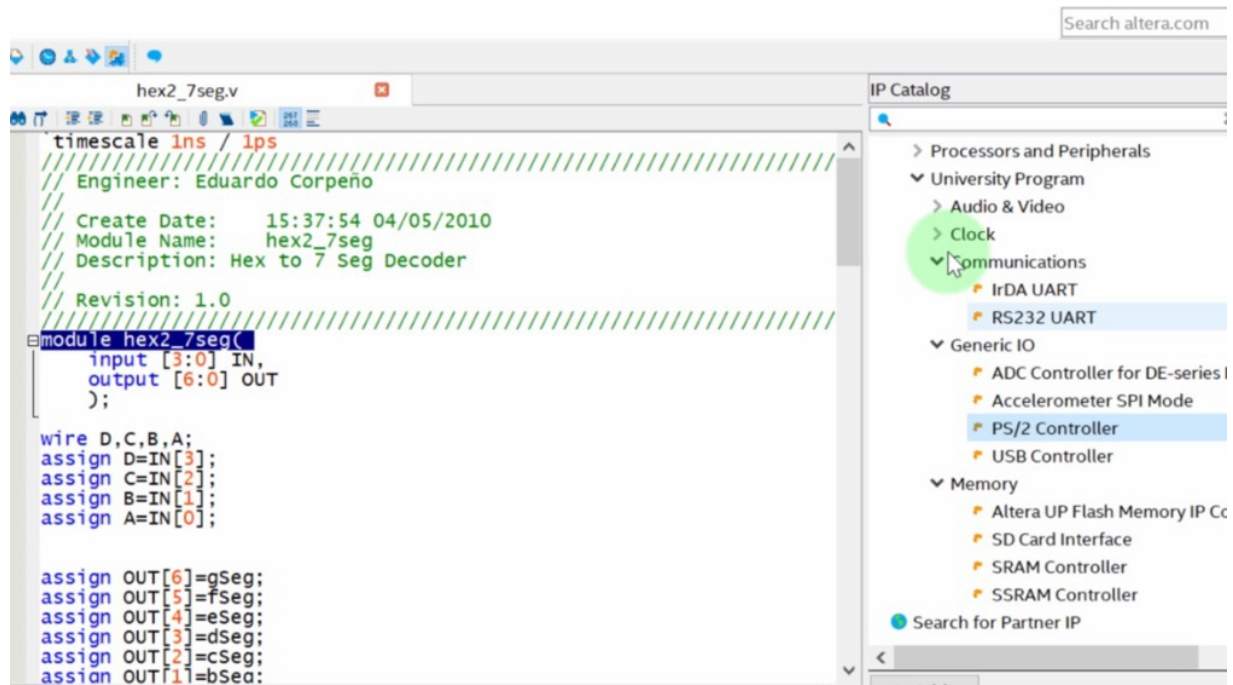


This part is called the project Navigator. And here we have a hierarchy view where I have my design and all of the modules it instantiates. Now here we have the tasks section. So here's the compiler tool chain. If we open this element we have all of the elements of the compiler. Here we have synthesis the fitter which is place and route. And this is the implementation actually. Then we

have the assembler which is the generator for the programming file. Here's the timing analysis part. Here's another step that Altera or intill takes. But beyond the compiler we have some other tasks. And the last one is the one we are most interested in finally which is program device. This part opens a separate application which is the programmer and we will see how to use this later. So these are the tasks to achieve completion. But there are several outputs or several purposes for which we can be using this application. And so we have some other biplanes we can use for example. We can implement a full design which takes more considerations and this is for implementing say a board of your own. So we are good with compilation for our project when we are working with a development board. But take a look at this. We have gate level simulation. So this is the gate level simulation. And the other is RTL simulation. Remember RTL is what goes in your source files very low or VHDL. So this is what we learned how to use. In the previous project in the series the one on very low when we used model same model same is one of many applications included in Quartus brain and we have another option here that says rapid precompile. This one only compiles or only goes through the steps that were affected by the changes you made to your design. So this one is to work faster because as you will see the compiled process is lengthy. It does take a while for my computer which is an ice 7 core seventh generation.



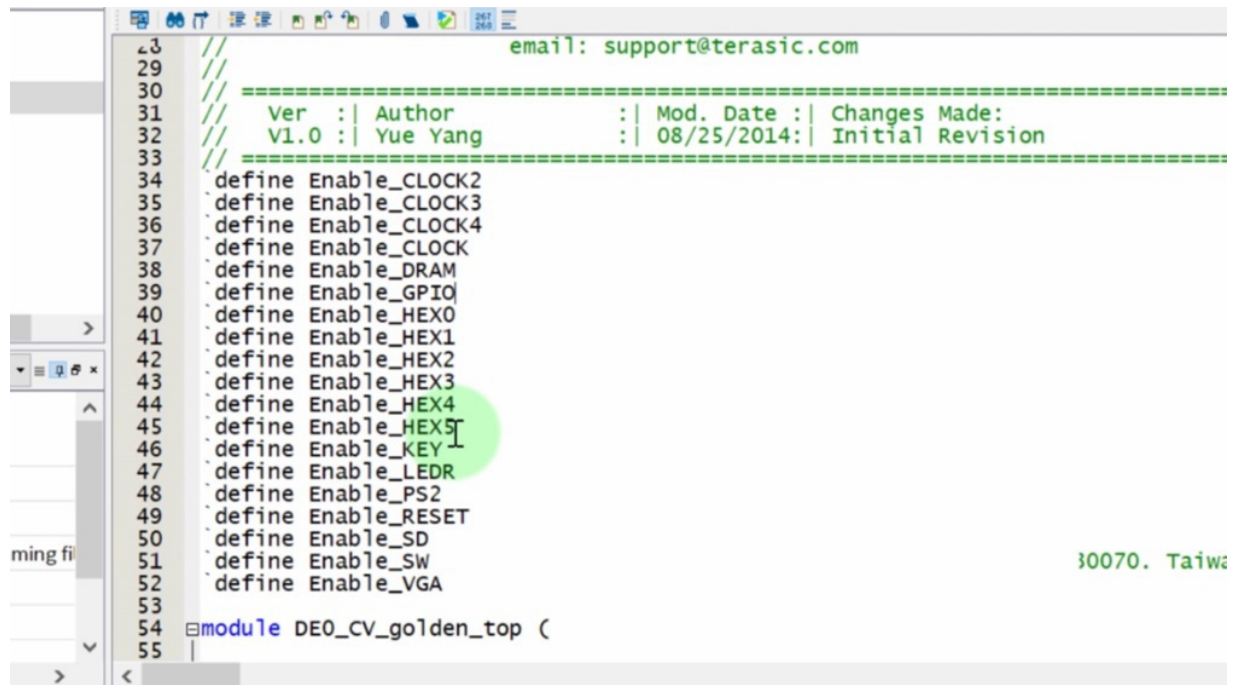
It's taking about 1 and a half minutes. So the bigger part of this idea is you're working section. If we don't click on some source file this is where we will see the code at the bottom we have a console and errors and warnings section. You may be already used to this if you have worked with any IDE. And here we have a very special part at the right which is the IP catalog. So this is a catalog of the AP projects that are available in Quartus so you can open this bar great here that says installed IP we have a project directory which has nothing but we do have a library here which has a basic functions. We have Arithmetike bridges and adapters you name it. Let's go to arithmetic and we have a lot of IP project which are models available for us to use and if we go to the very bottom we can see the university program I.P project here. So these are for education. And here we have for example generic input output and you can see that we have a U.S. controller we have a p S2 controller remembered that the Ds 0 board does have a s2 connector. Well maybe you can use this one for the board.



We have communications for example we have and RS232 which is a serial port. We have memory here. So here's the SD card interface which may also work for our board and we have a memory controller which again our board does have some SRAM. So all of these IP projects are ready to be used if you want to and maybe we'll do a design on this in a later project on somewhat more advanced designs. But I wanted to mention it because this is a big motivation to start working with real professional applications with already proven Cores. There are many many more parts of the FPGA design process that you will learn along the way. But for now this is Quartus.

LOOKING AT THE TOP-LEVEL TEMPLATE CODE

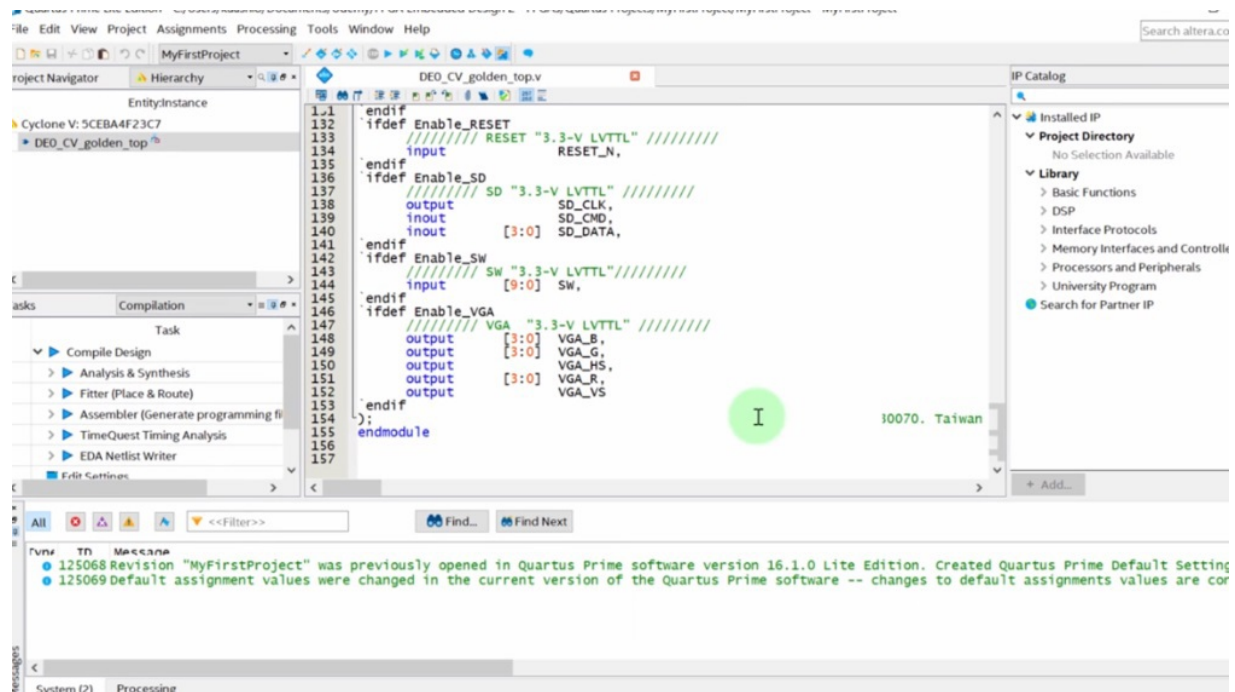
Now if we take a look at it it's very low code. Let me show you here we have a lot of definitions for symbols. Remember the pound define equivalent in very low uses not the pound symbol but the rather unique grave accent character.



```
23 // email: support@terasic.com
29 //
30 // =====
31 // Ver   :| Author           :| Mod. Date :| Changes Made:
32 // V1.0 :| Yue Yang         :| 08/25/2014:| Initial Revision
33 // =====
34 define Enable_CLOCK2
35 define Enable_CLOCK3
36 define Enable_CLOCK4
37 define Enable_CLOCK
38 define Enable_DRAM
39 define Enable_GPIO
40 define Enable_HEX0
41 define Enable_HEX1
42 define Enable_HEX2
43 define Enable_HEX3
44 define Enable_HEX4
45 define Enable_HEX5
46 define Enable_KEY
47 define Enable_LED
48 define Enable_PS2
49 define Enable_RESET
50 define Enable_SD
51 define Enable_SW
52 define Enable_VGA
53
54 module DE0_CV_golden_top (
55
```

So anyway here is a symbol defined that is called and Nabl clock to then enable clock three four o'clock DRAM G.P.O. hex from 0 to 5 d l d r and all of these symbols are used to create a new Verilog module called these zero C-v gold and talk. This is your design and the input and output ports are included as you can see here only when their flags are included. So this is conditional inclusion of code as in C or C++. And this is where the definition and it's. And you can see a nice

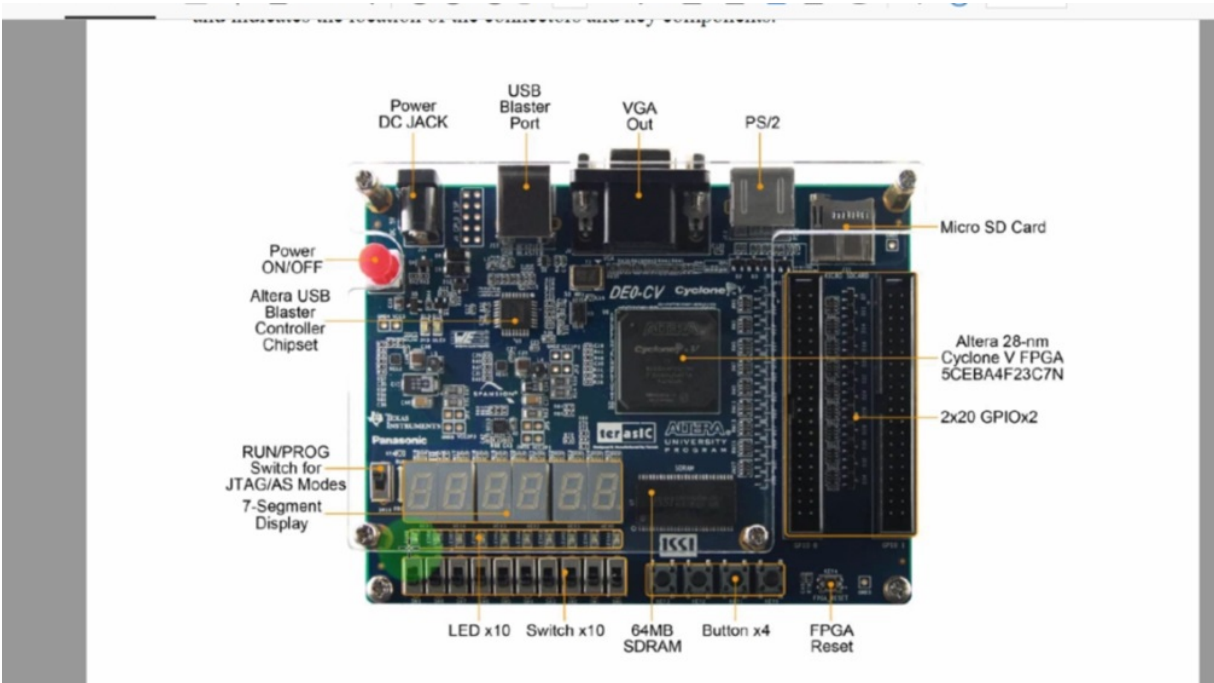
bug from Quartus right here. It left a piece of text where it shouldn't be oh well anyway.



We can use any of these lines that correspond to the lines in the hardware. So for example the lines that say G.P.A. or here are from 35 to 0 in G.P.O. 0 and from 35 to 0 in G.P.O. 1 These are input output pins. That's why they are called inout they are bidirectional. Here we have for example the LCD array It's only called LCDR and it's from 9 to zero. So there are 10 Leds the keys the four buttons are here from 3 to 0. This is an array of inputs. And so as you can see these are all of the lines we care about.

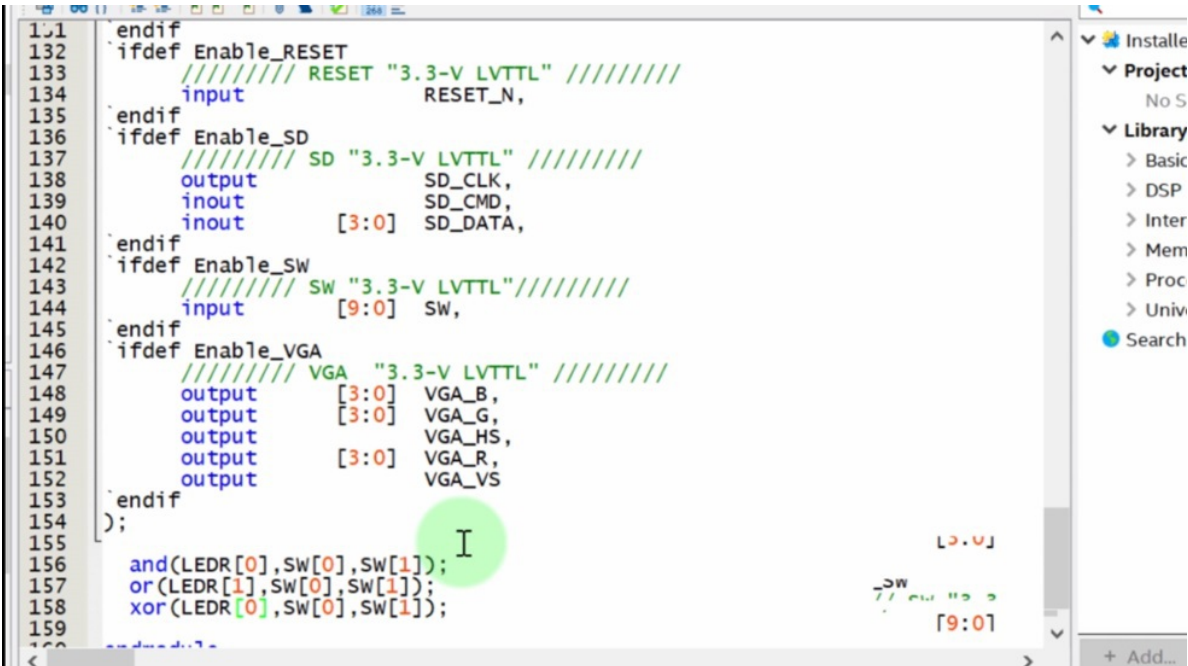
ENTERING SOME PROOF-OF-CONCEPT CODE

So let's make a proof of concept application right now. Let's take these two switches which are switch 0 and 1 and perform an AND operation. So let's instantiate an AND gate and put it in this Elodie number zero then or again right here for the same two inputs and just for fun. Let's use these two switches right here.



These must be 8 and 9 and put there x or outputs in the last LCD which should be 9 back to Quartus. You may already know this takes only three lines of code. First the end gate that will send its output to Aley the R orders zero and the inputs will be switch as w 0 and

switch 1. I know these names but here they are. Here's the switch array and the array is somewhere here. OK.

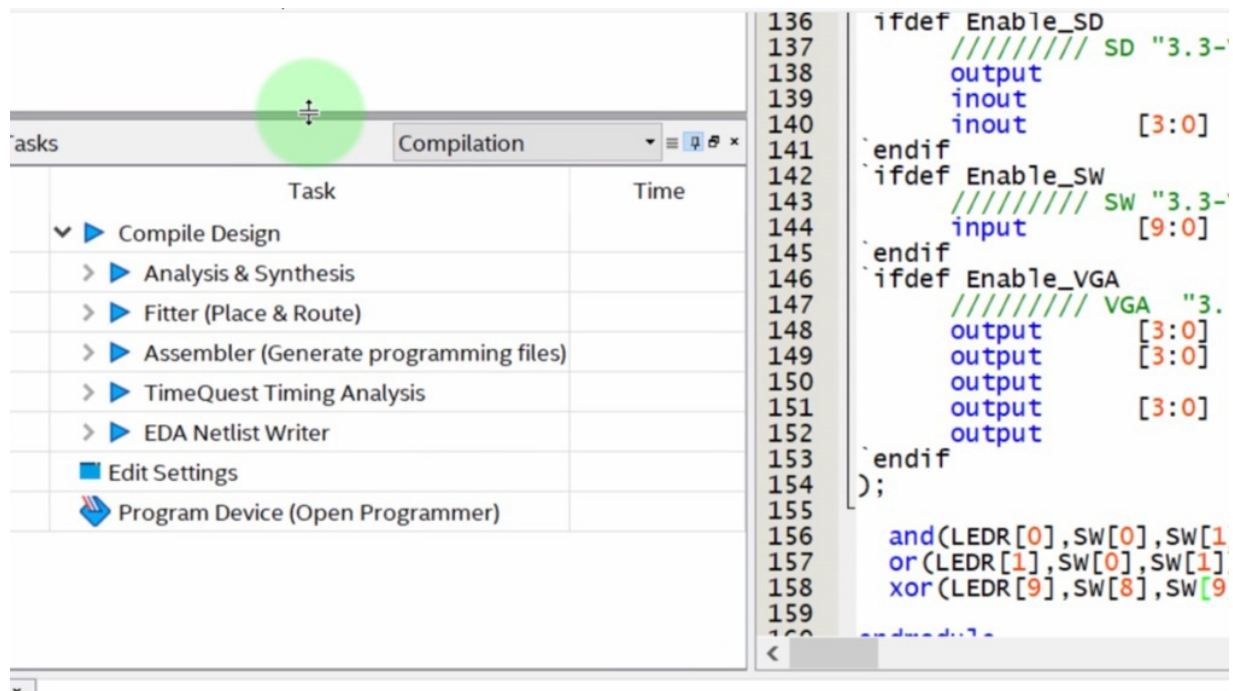


```
131 .endif
132 .ifndef Enable_RESET
133     //////////////// RESET "3.3-V LVTTL" ////////////////
134     input          RESET_N,
135 .endif
136 .ifndef Enable_SD
137     //////////////// SD "3.3-V LVTTL" ////////////////
138     output         SD_CLK,
139     inout          SD_CMD,
140     inout          [3:0] SD_DATA,
141 .endif
142 .ifndef Enable_SW
143     //////////////// SW "3.3-V LVTTL" ////////////////
144     input          [9:0] SW,
145 .endif
146 .ifndef Enable_VGA
147     //////////////// VGA "3.3-V LVTTL" ////////////////
148     output         [3:0] VGA_B,
149     output         [3:0] VGA_G,
150     output         VGA_HS,
151     output         [3:0] VGA_R,
152     output         VGA_VS
153 .endif
154 );
155
156 and(LED0[0], SW[0], SW[1]);
157 or(LED0[1], SW[0], SW[1]);
158 xor(LED0[0], SW[0], SW[1]);
159
160 .endmodule
```

Now let me copy and paste this line twice more and I said we were going to make an OR gate and an X or a date. So the output for the door will be an LCD one and the output for the X or will be an LCD 9 but its inputs will be switch 8 and switch 9. That's it. This is my application.

COMPILING YOUR DESIGN

So let's compile and inquire. You have all of your tasks in this separate section right here and label task. And so we have the compiled design toolchain right here and we have several steps. One is called analysis and synthesis. This is the equivalent to the compilation process. Next we have the fitter which is classically known as place and route. And what this does is find the best layout for the actual hardware inside the chip. This is very advanced stuff. Next we have the assembler.



The screenshot displays a software interface with two main panels. On the left, a 'tasks' panel is visible, featuring a table with columns for 'Task' and 'Time'. A green circle highlights a plus sign icon above the table. The table lists several tasks under a 'Compile Design' category:

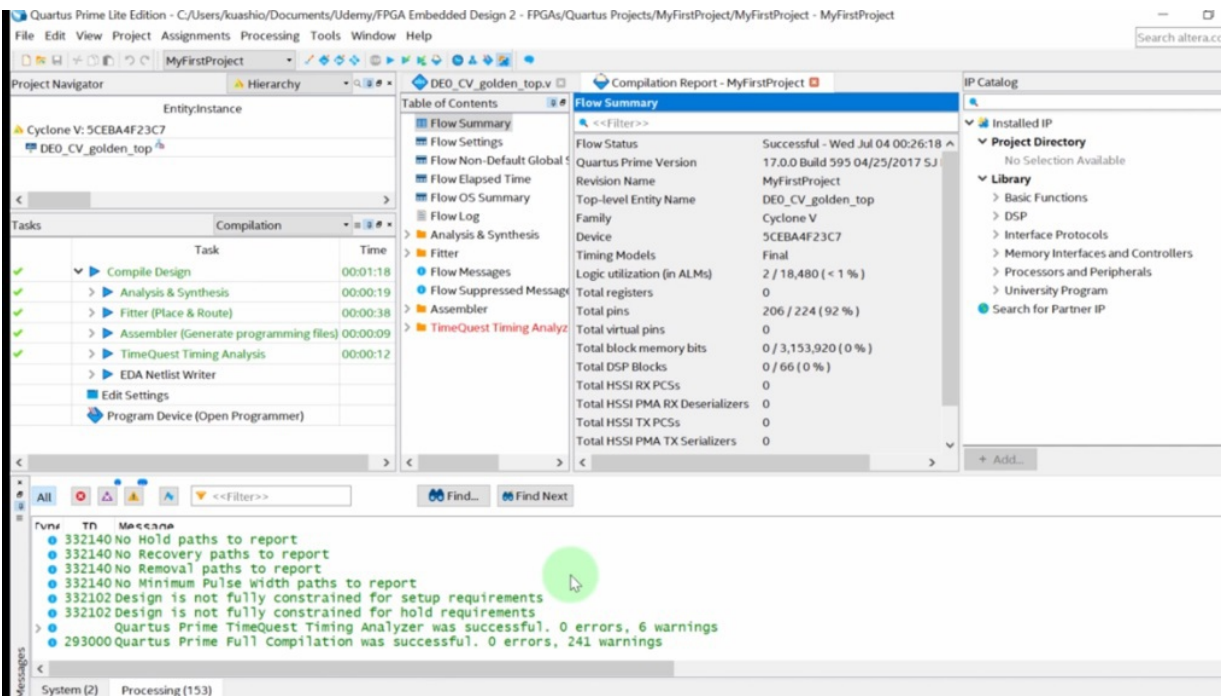
| Task | Time |
|----------------------------------------|------|
| Compile Design | |
| Analysis & Synthesis | |
| Fitter (Place & Route) | |
| Assembler (Generate programming files) | |
| TimeQuest Timing Analysis | |
| EDA Netlist Writer | |
| Edit Settings | |
| Program Device (Open Programmer) | |

On the right, a code editor shows Verilog code with line numbers 136 through 160. The code includes conditional compilation directives and logic for LEDR, SW, and VGA signals:

```
136   ifdef Enable_SD
137       ////////////// SD "3.3-
138       output
139       inout
140       inout          [3:0]
141   `endif
142   `ifdef Enable_SW
143       ////////////// SW "3.3-
144       input          [9:0]
145   `endif
146   `ifdef Enable_VGA
147       ////////////// VGA "3.
148       output         [3:0]
149       output         [3:0]
150       output
151       output         [3:0]
152       output
153   `endif
154   );
155
156   and(LEDOR[0], SW[0], SW[1]
157   or(LEDOR[1], SW[0], SW[1]
158   xor(LEDOR[9], SW[8], SW[9]
159
160   and(dma[1]
```

This part is similar to an assembler for code for microprocessor code that takes all of the output files and puts them all together and

generates a programming file. The file that we will ultimately download into the board. These are timing constraints this time quisque timing analysis helps us comply with the requirements of timing in our designs for our examples. We will not pay much attention to it but this is very necessary. Once you get the hang of this this is the next step. You need to inform your development platform about the frequency you expect your design to run at. So right at the bottom we have the programmed device task. This one actually opens another application that is a programmer which will be our next step after compiling. One more thing about these tasks is that they all have subtasks and inside they have several other tasks and useful tools. You may want to take a look at at some point.

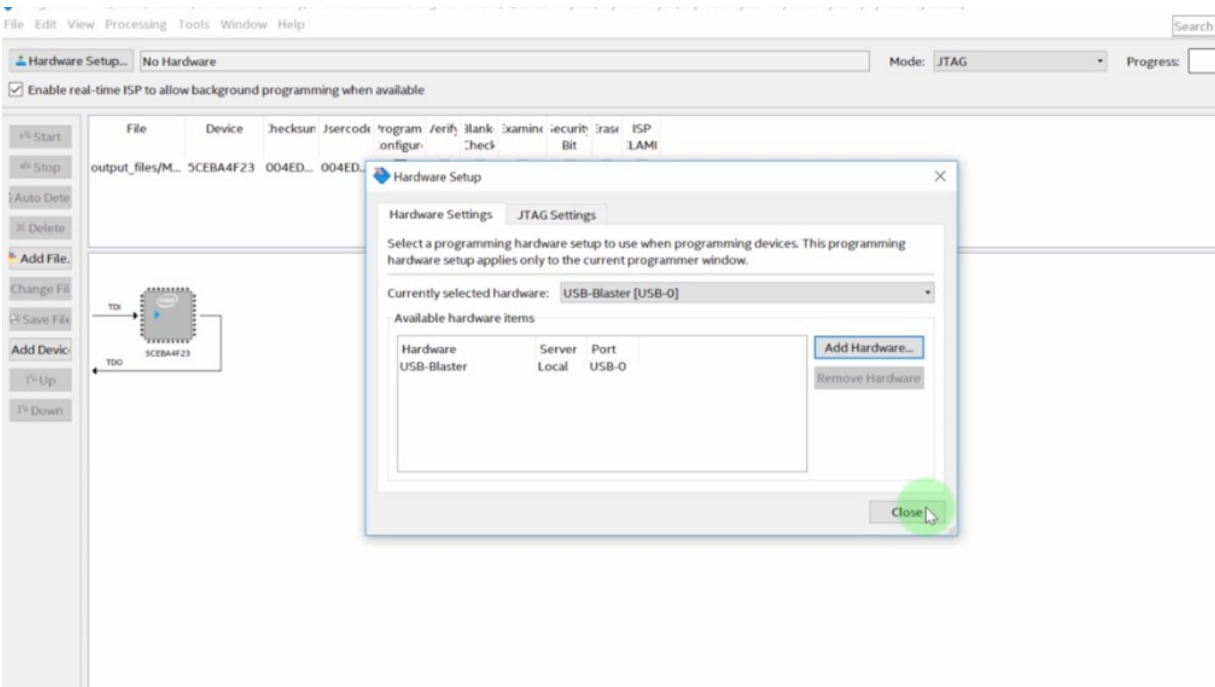


All right so let's double click on compile design yes I do want to save my changes. And this will take a while. At the bottom we get the classical console window which will show us warnings and eventual errors. Some of the warnings we will get are for unconvicted hardware because our application is not using much of it anyway and some others are for the timing constraints which may not have been met because they haven't specified anything about time. This is just a combinational application with three gates. It's still working.

As you can see some of the tasks are done yet and we are finally done. So if you get this message Quartus brain full compilation was successful 0 errors you are golden. You're not so golden if you get so many warnings so you may want to take a look at the first warnings and then make sure everything's OK. For our design Trust me this is OK.

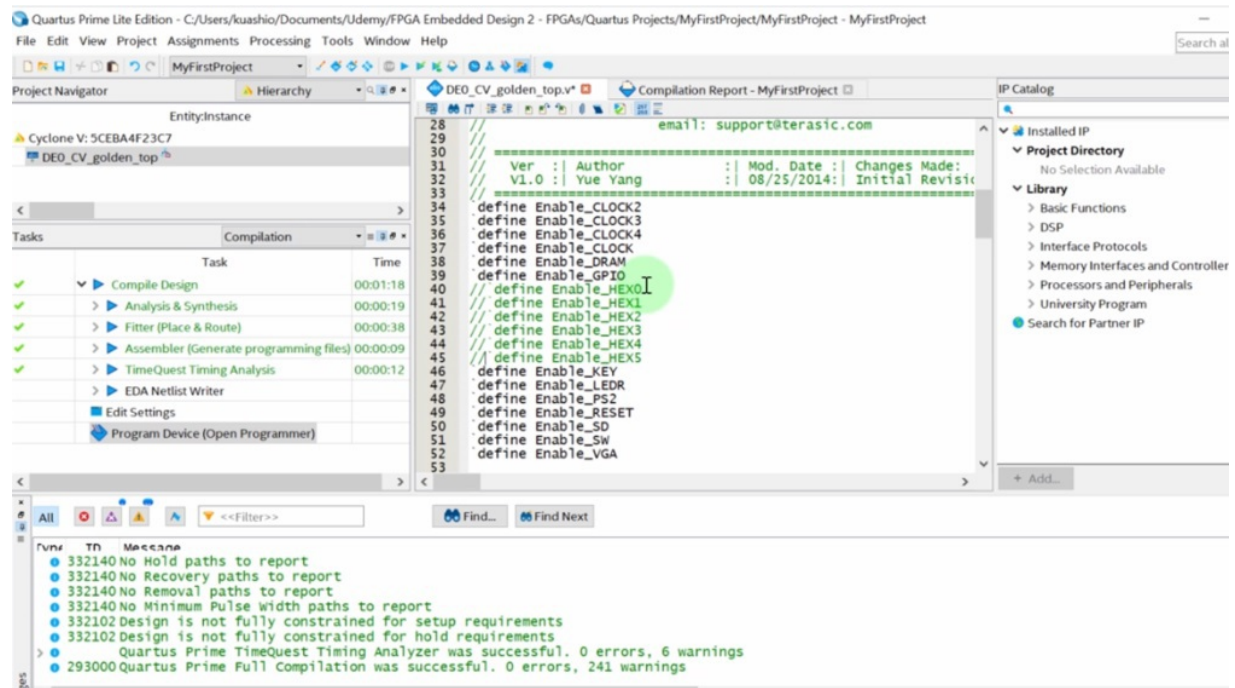
PROGRAMMING YOUR DEVICE - JTAG MODE

All right. So now we don't click on program device once more this will open another application. This is the programmer. And as you can see in this part right here it says that there is no hardware connected to it. And there's a good reason for it. There is no hardware connected to it. So let me connect the board and if you look at it it's running the Demel application that was loaded into it by the manufacturer. And now let me press here. Hardware setup and choose the hardware that's connected through the JTA connector.



And so if you choose right here you will see you as the blaster you will be zero. And this is done immediately. So I just close here and I am free to press this start button. You have to make sure that your

board has the run programming switch in the run position the top position. So this may tell you that we are going to program the FPGA. So we need that. So we're fine. Let me start as you can see the board is seemingly doing something and now it's done so a kind of annoying thing here. Are all of those Ltd's in the displays that are on. So let me go back to the design and just turn them off. An easy way to do that is going back to the code and disabling all of those hex displays. And so we can go starting at line 40 just ignoring commenting out these definitions.

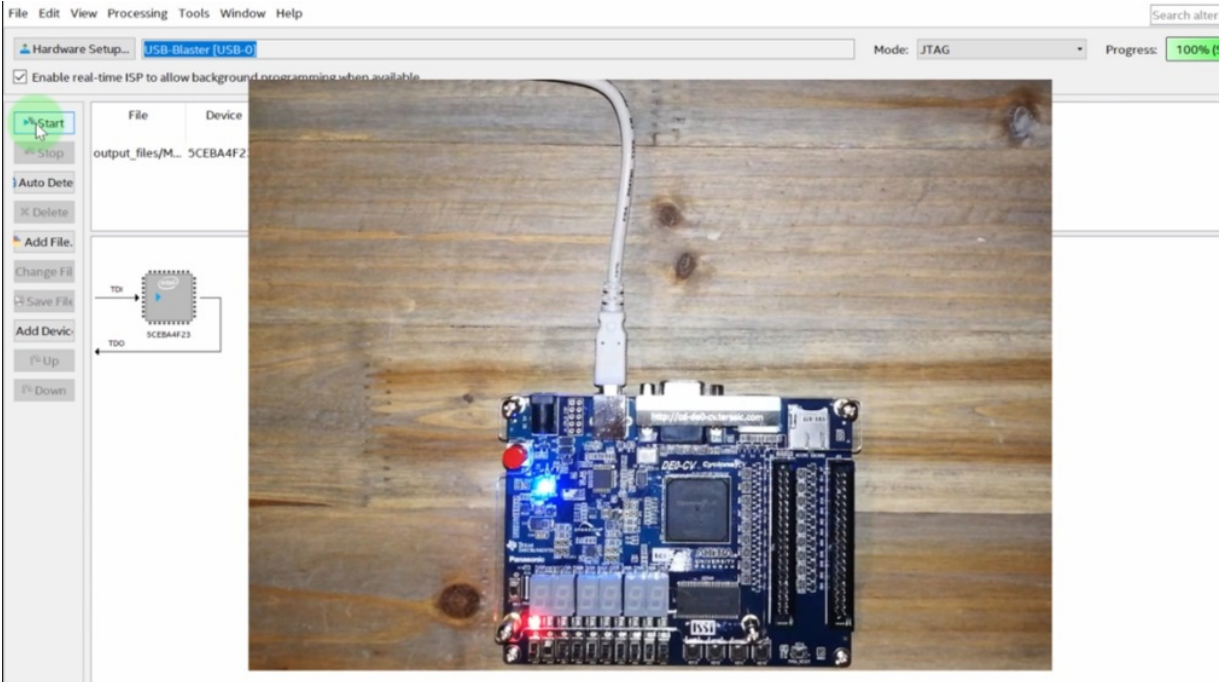


If you come in all of those out these outputs will not be used. And so the displays will not be driven so they will be off. Let me compile again. And now that it's done let's open the programmer once more as you can see it kept its configuration. So we no longer have to look for the hardware let me start and this you can see now displace all Ross. So let me manipulate the board and show you how the lower order switches are indeed working as an OR gate in the second LCD from the right I will turn just one of them on and the second Elizee lights up I will turn them both off and both Ltds are off. I'll turn them both on and Poles Ltds are on and they'll turn just one of them on as

you can see the first lady is an AND gate and the second is an OR gate.



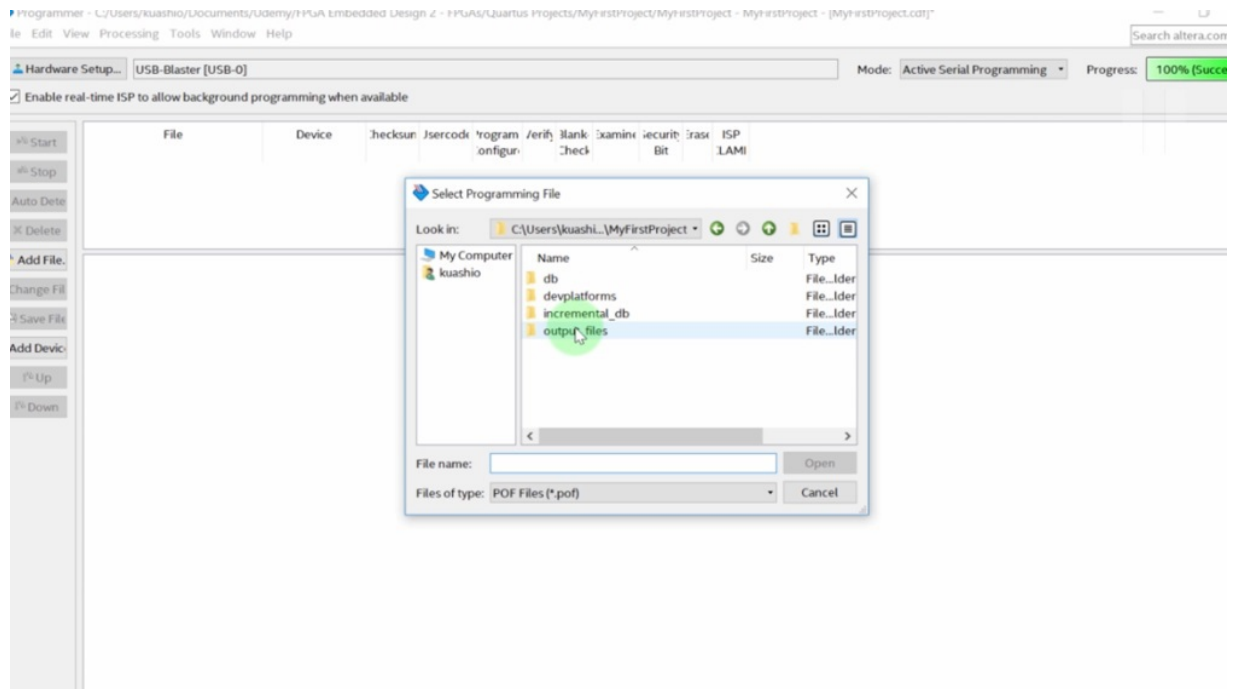
And what about the left most switches. Well these are configured in an X or fashion with the left most Ltd. So these are both off. Let me turn just one on and the radio goes on. Let me turn the other one on. The LCD goes off and let me turn the other one off again. And the turns on no one important thing about the way I downloaded this application to the board is that I loaded it into the SBA into the static ram of the FPGA. So when I turned the power off it will lose all of its hardware configuration.



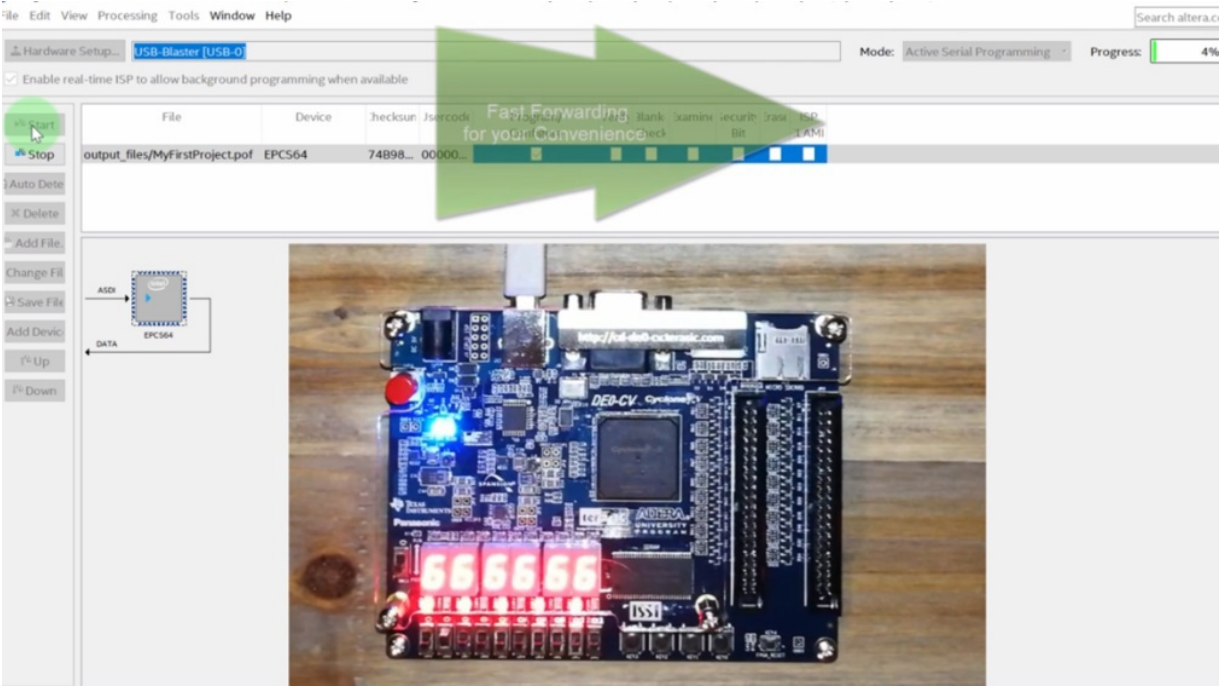
And by the way I am not using the DC adapter. I have simply connected the U.S. B cable to the board that will provide power to the board and well the manual says that you have to always use the DC adapter but for this demo we're OK with just us me cable. OK let me turn off the board and now turn it back on see theres the default application again. And so next let me show you how to download the same application but now to the nonvolatile memory to the serial configuration device.

PROGRAMMING YOUR DEVICE - ACTIVE SERIAL MODE

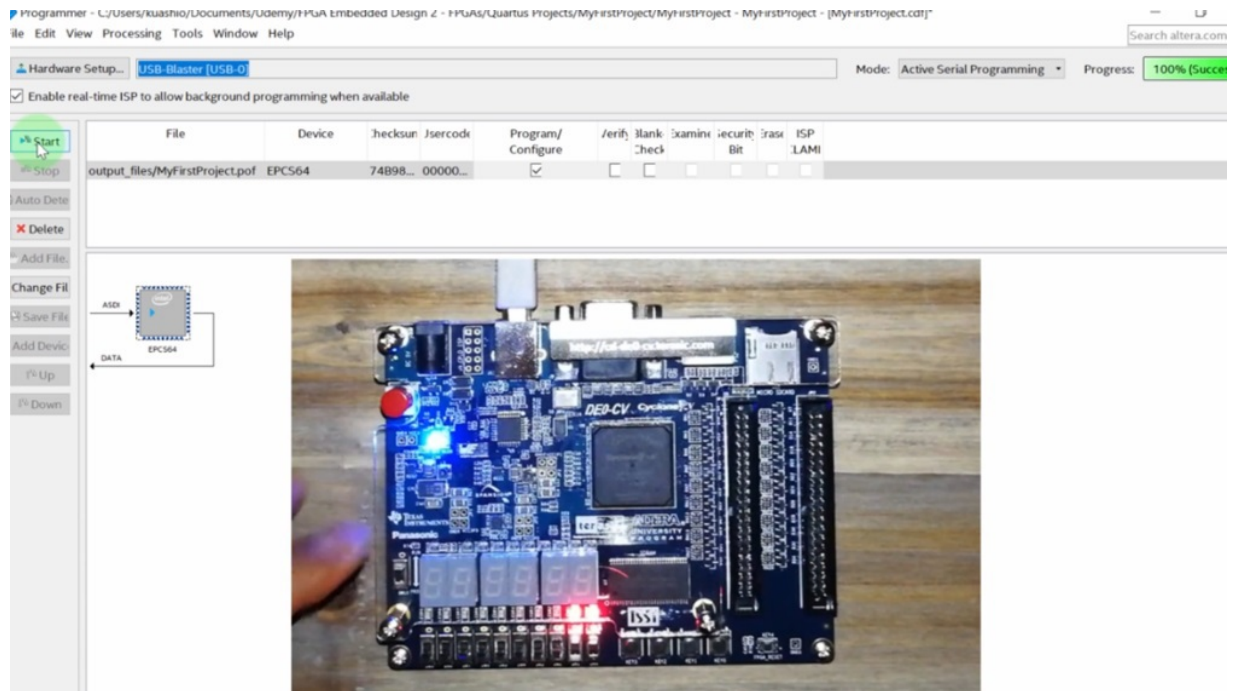
So to send the application to the serial configuration device the nonvolatile memory. First we have to move the switch. The run program switch to the programming position. That's the first thing you should do next. You can go right back to the programmer application and just change the connection. You can change it right here where it says mode this modus JTA. That's where the figure is connected but you can change it to active serial programming. If we go there we get a warning that the configuration isn't consistent with the mode. So let's make it consistent. We are supposed to have a file here the file we were supposed to send here wasn't the S.O.S. But the POS file.



So these are in the output files folder for our design. And here it is deal with. So when they choose it I have to check on the first option right here that this program configure. I always have to do this even for the FPGA. So let me hit start here. And this process takes way longer than just programming the FPGA. For some reason programming the flash memory in the nonvolatile chip takes much much longer. No don't pay much attention to that progress bar. It's not uniform. At some point it will just reach 100 in one step.



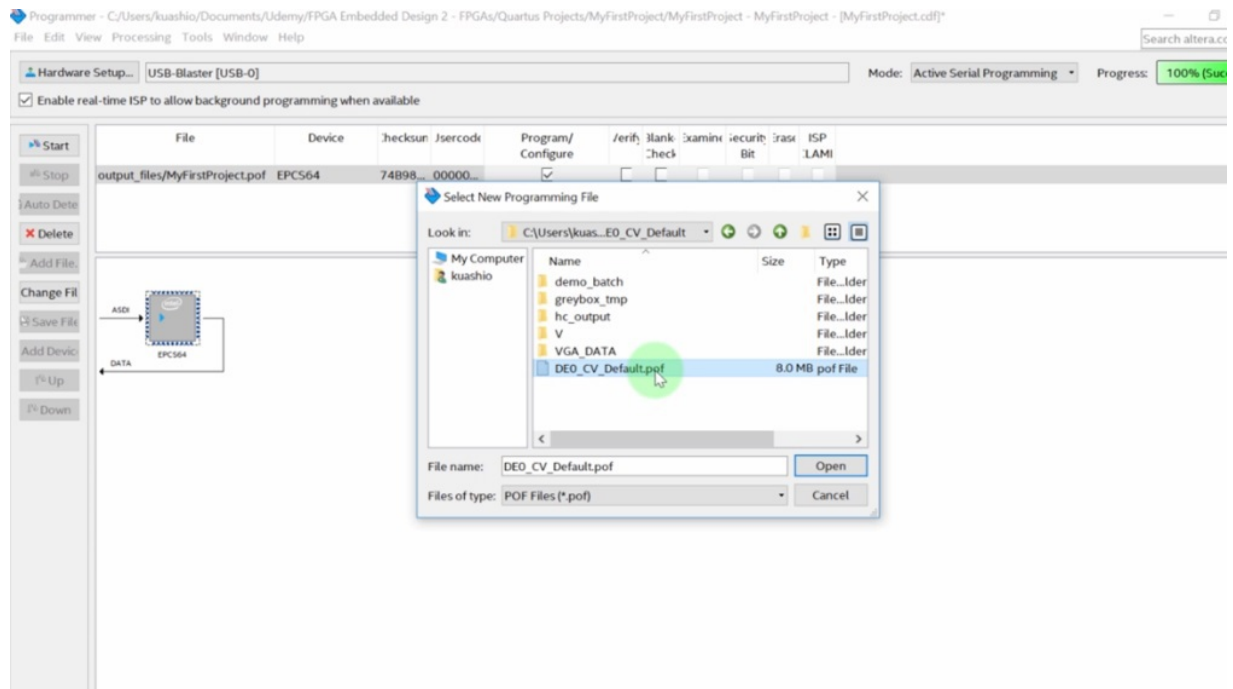
There it is. So now if we look at the board it's still running. The default application. But that's because they haven't cycled the power. So let me do that. Let me turn it off while the board is off. Let me move the switch back to run now let me turn it on again. And as you can see it is working. Passing and and or gate in the right most switches. And as an extra gate at the left most switches.



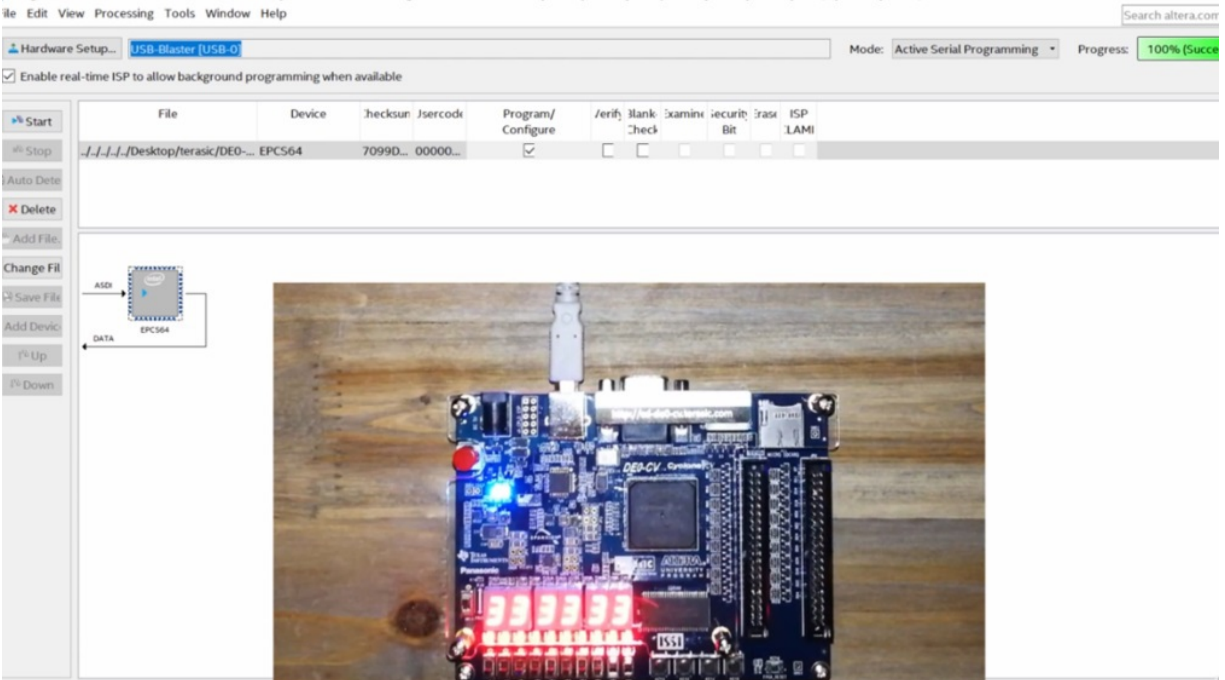
So you should use this method when you when your application is to survive a power cycle. For example if you want to show your application to someone and you have to turn off the board then you should do this. It takes longer to program but it's nonvolatile. And this is the way most FPGA work. They download their configuration on boot up. This is not some quick fix but really an actual method of using FPGA is.

GETTING IT BACK TO ITS FACTORY STATE

And if for some reason you want to restore the factory application that was loaded in the board when you first unboxed that you can do it so you can find that one if you select the file.



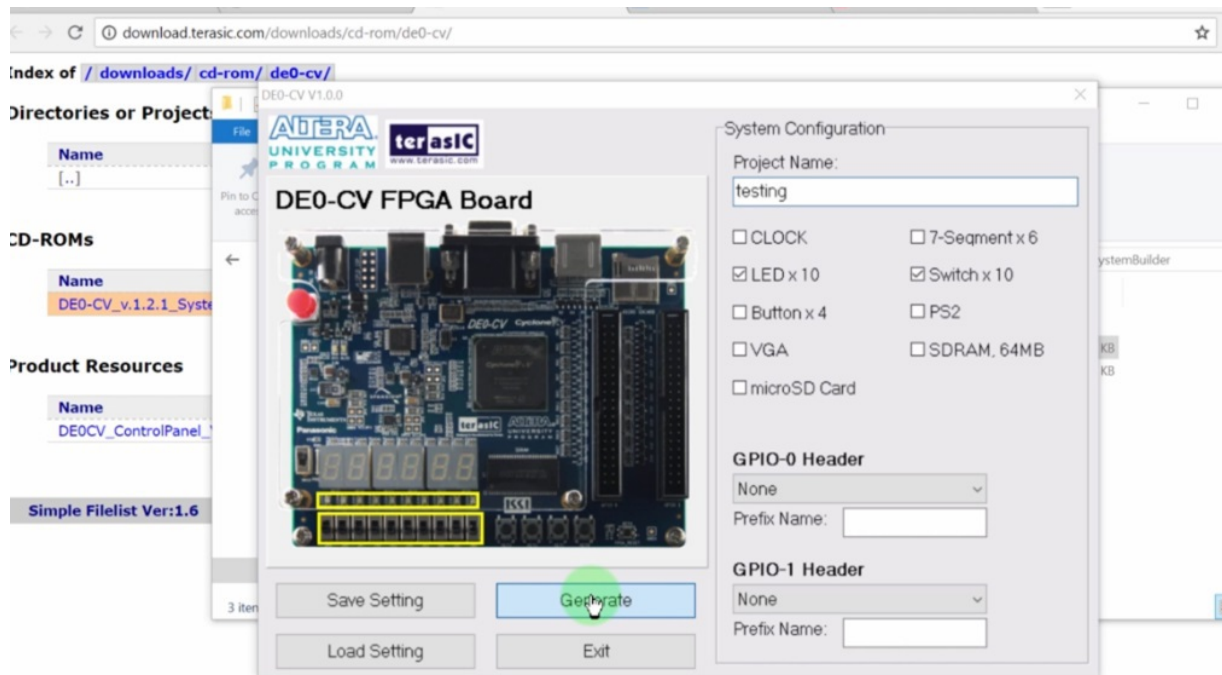
I will double click on the file and you can go to your directory where you downloaded the CD-ROM content. Let me do that here. So here you can go to demonstrations then the zero C-v default and here is the deal with file. Let me double click on it. Check on programs configure and before hitting starts. I have to move the switch to the program concision and now I can hit start once again it's taking long to do it. But you have to be patient.



It's been a while now. It should be almost done. There is a and so let's turn the board off. Then change it to the RUN position and finally turn the board on again. And there it is. There's a more obligation so you can see there was no harm done. Your board is back to its brand new state.

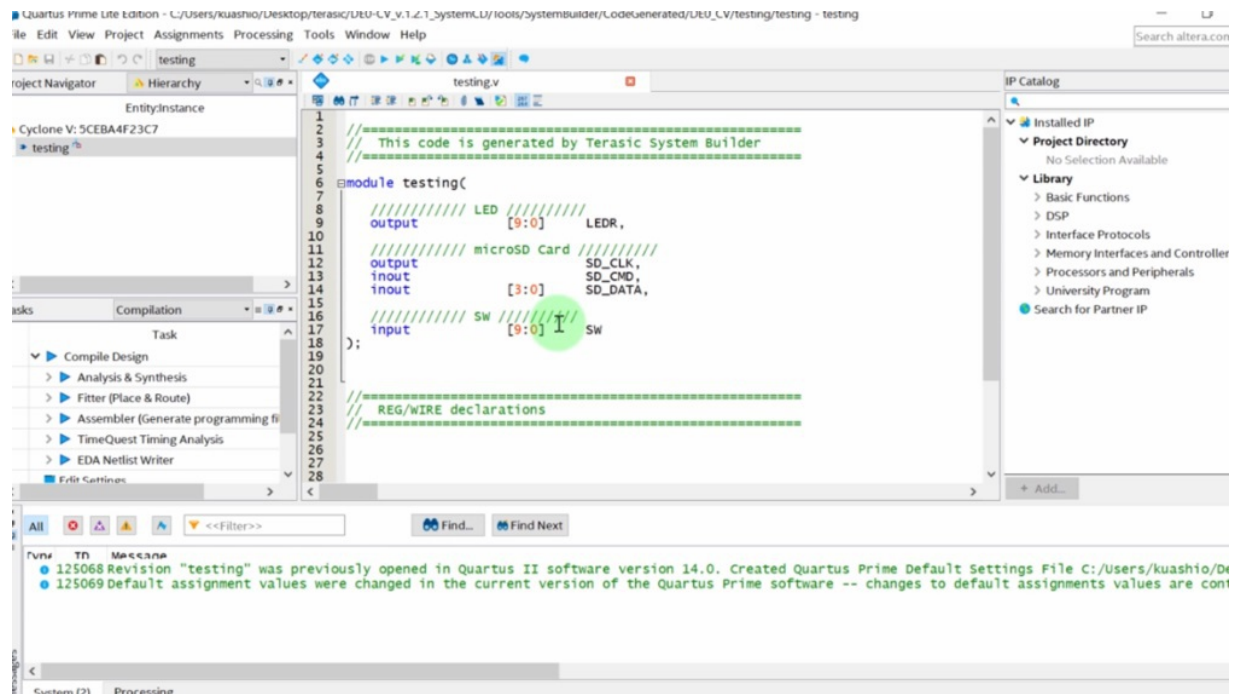
SYSTEM BUILDER_ THE EASIEST WAY TO JUMPSTART YOUR APPLICATIONS

And here's another two named system builder. So with this one you can create your own projects by selecting which elements you want to use. Then breasting generate to generate your projects. And so let's give it a try.

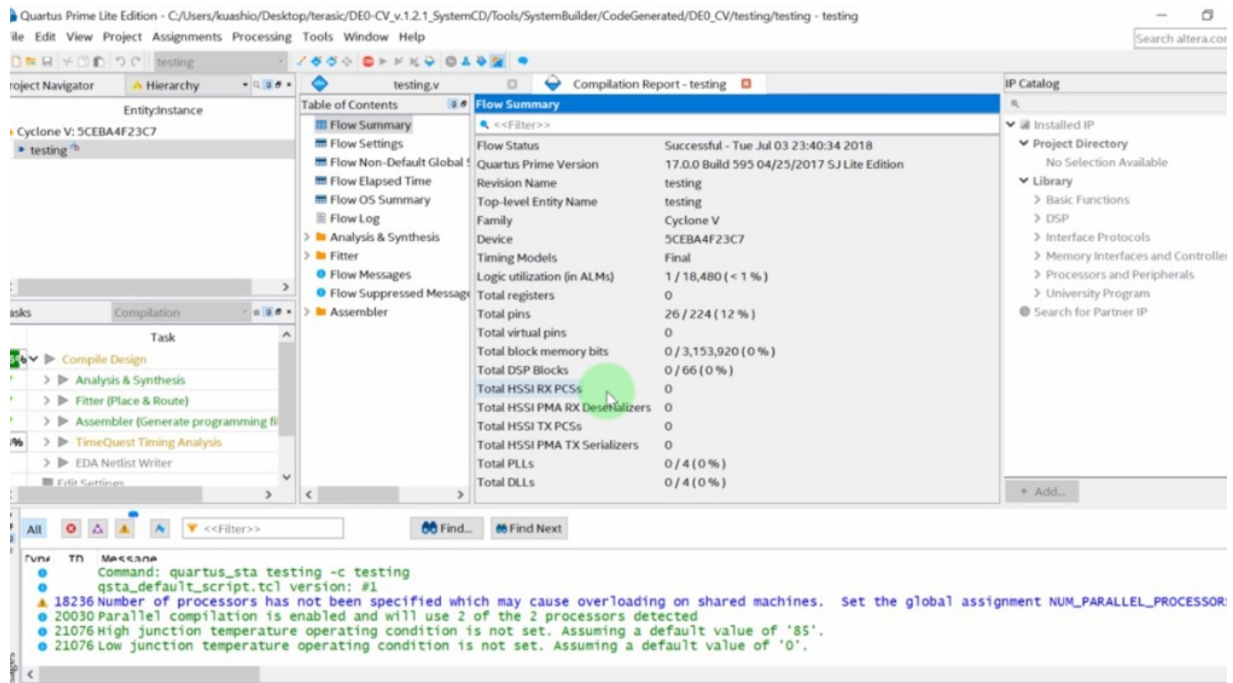


Let's say we want to use only the LEDs and the buttons no switches that say we only want to use LEDs and the switches so let's generate the project. And here you can name your project. So let's

say testing let's generate it. It appears to have been successful. And so that's exit and check the generated code. Here's the testing folder. And here is that B.F. file for Quartus Let's see what happens if you look at the testing module right here.



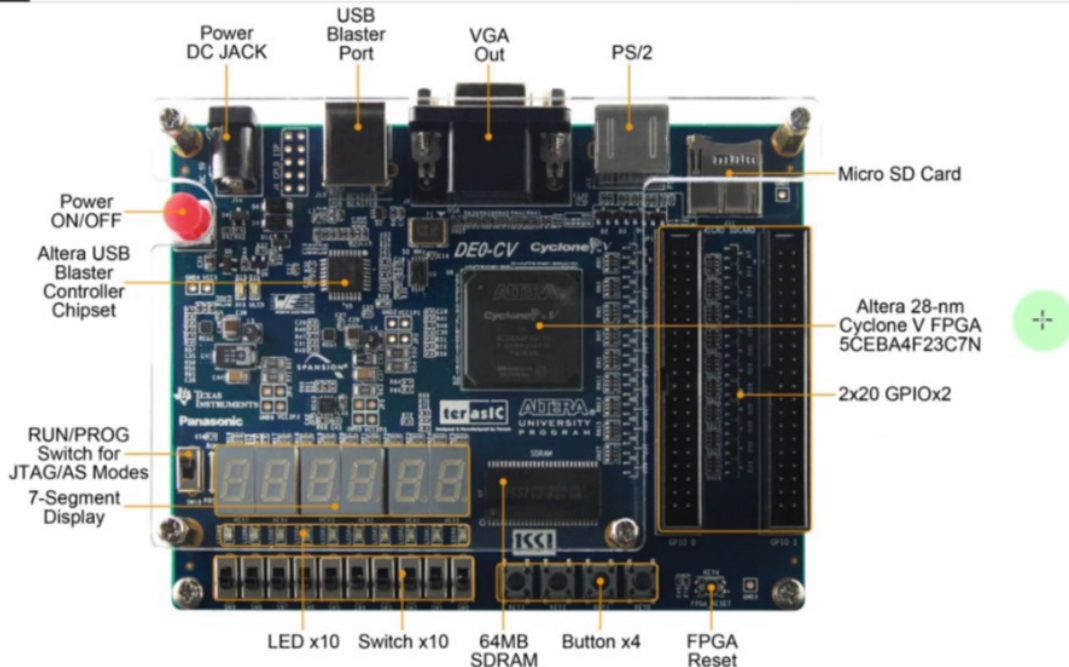
It only has the lead the array of 10 Leds. This is very long by the way it has the SD card elements. I'm not sure why and the switch inputs. That's it. Here you can write your code and maybe try it. So let's see. Let's just as an AND gate here. Which makes sense it's output to one of the led's that say led are number four and takes in switch numbers 0 and switch number 1.



That's it for this design. Let's hit compile this. Now that it's done let's try and download this application into the board and see what happens. Let's move switches 0 and 1. And there it is. It's working. So yeah this is a very nice way to start up your applications. You can jumpstart your applications by using this nice system builder once more. This tool is under Tools Folder. It's called System Builder. You just run this executable file and it will build an application for you.

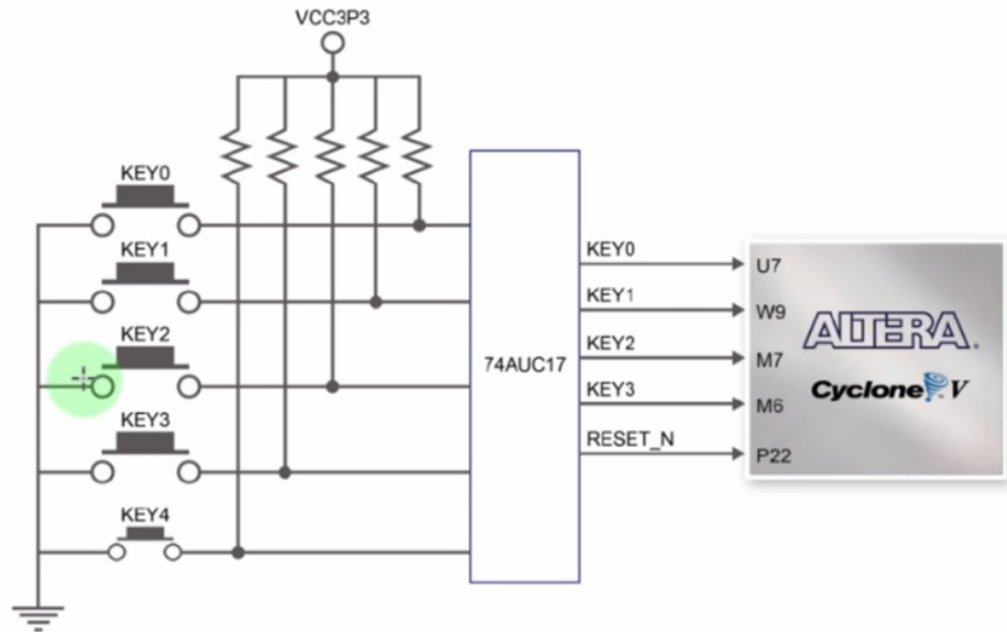
SYSTEM DESCRIPTION

Now let's make an either the same forbit either we decided earlier in the very long project. And let me show you how I plan to use the board. Let me use the picture we have in the User Manual. So I will use this to displace the two in the middle to show the two variables we are going to have.



So these are four bit numbers so these fit in hexadecimal digit each. And so the parameter a will go here b we'll go here and the addition the sum of these two will go into these two displays. That's because the highest number we will get in this other will be from F plus F which is 1 e. And so I will print right here either a 0 or 1. And here a number from 0 to f. Let me enter a and b with the switches. Let's do this at the right most treacherous so switches from 0 to 3 will be the upper end B and switches from 4 through 7 will be the upper end. A.

So the way I will control this is that I will enter a here and it will appear in this display and I say change. Be here. It will be reflected in this display. And finally the result from the other that will be inside the FPGA will go to these lines. And just for kicks I will show you how to use the oscillator in the board. So I will get the 50 megahertz signal that is onboard. I will make a clock divider model in very long and use it to show in the least significant LCD a blinking of either 240 hertz or one Hertz. I will switch between these frequencies using the least significant of these buttons. That's the rightmost one. So when I press this button the LCD will blink very fast at 240 Hertz but when they release it it will blink at 1 hertz. So once more I will implement the forbit aether and also a blinky application on this LCD which will have its frequency coming from the internal oscillator of the board. Now there are some things we need to know. First we need to know how to control the displays. So we need to know if a zero or a one turns the segments on and we need to know if they are connected in a bus because if they are we will need a restor. So we need to turn one on show its value then turn it off then turn the next one on because when they are connected in a bus you cant turn them on at once with different values. They would show the same thing. But luckily this board has each of these displays connected exclusively to the FPGA. We also need to know if the Mideast are on or off with ones or zeros and what the buttons and switches send into the FPGA at each state. So presumably having the switches down or the buttons depressed is a zero. And having the buttons pressed or the switches up you see one but we'll have to verify this in this user manual. So lets just make sure about these details. So here at page 23 we have the circuitry for the tese.



And as you can see pressing them sent in a zero logical zero. So I was wrong when I said that Bressant them might have been a one. It's not it's a zero and it has pulled up resistors right here all right. So next we need to know about these switches. And here it is putting them up is a logical one and down to a logical zero. So my suspicion was correct here.

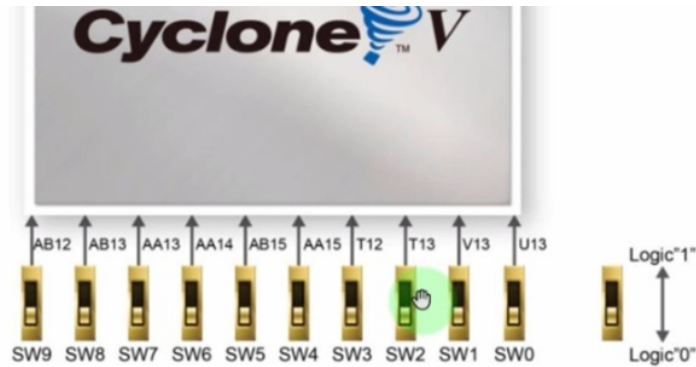


Figure 3-7 Connections between the slide switches and Cyclone V FPGA

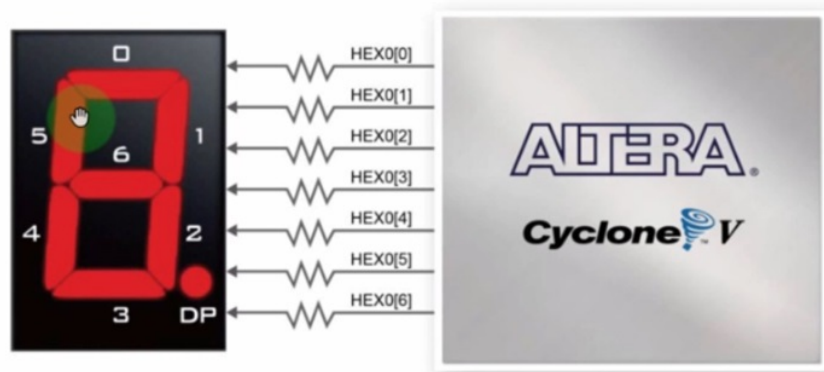
■ User-Defined LEDs

There are also ten user-controllable LEDs connected to FPGA on the board. Each LED is driven directly by a pin on the Cyclone V FPGA; driving its associated pin to a high logic level turns the LED on and driving it to a low logic level turns it off. Figure 3-8 shows the connection between LED and

Next we have the ladies and as you can see they are driven by their anodes and the cathodes are connected to ground. So yes they are turned on by logical ones and last but not least we have the display. So yes each display has one line dedicated to each of its segments in the traditional order 0 1 2 3 4 5 and 6 which are classically named A B C D E F and G. Apparently you don't have control over the decimal point.

The DE0-CV board has six 7-segment displays. These displays are paired to display number various sizes. **Figure 3-9** shows the connection of seven segments (common anode) to pins Cyclone V FPGA. The segment can be turned on or off by applying a low logic level or high level from the FPGA, respectively.

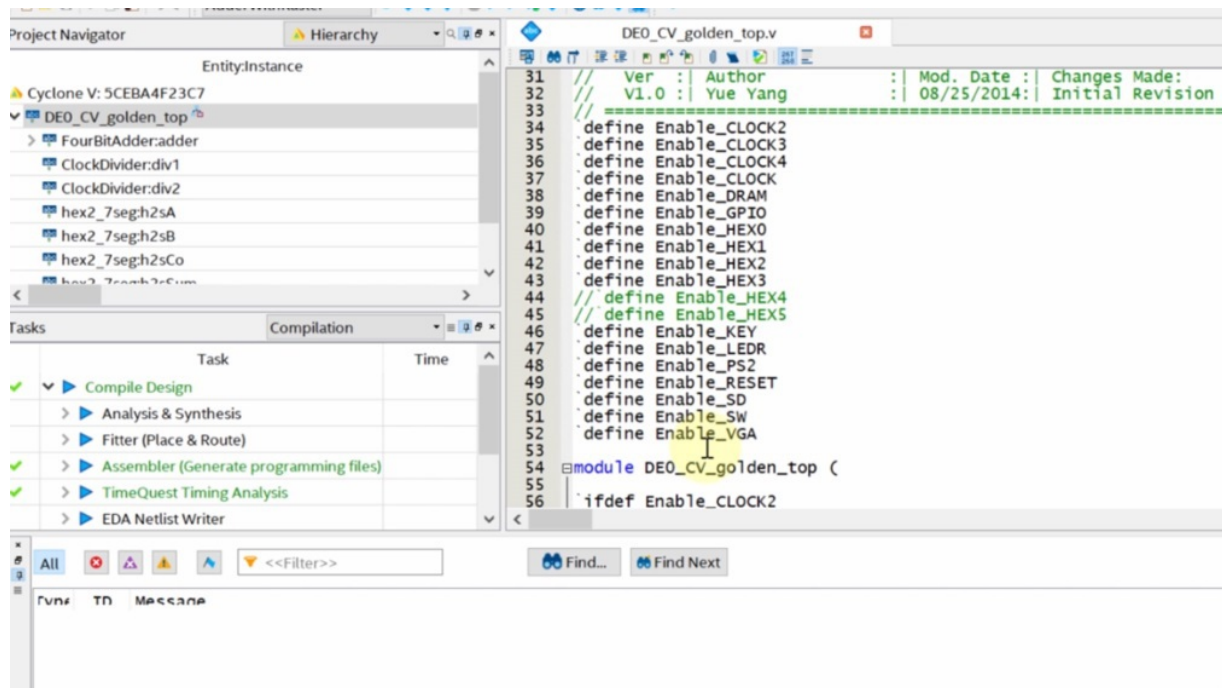
Each segment in a display is indexed from 0 to 6, with corresponding positions given in **Figure . Table 3-5** shows the pin assignment of FPGA to the 7-segment displays.



And somewhere in the manual it says that they are come on a.. Which means that a logical zero turns on the segment and D.A. is presumably connected to the logical one line or the supply line. You can always make sure about this by looking at the schematic diagram of the board. If it's very important to you. And why not. Let's take a look. So browsing through the schematic let me just find the seven segment displays. And here they are as you can see the come on and nodes are all connected to a VCC line which means a logical one. All right. That's it. So let's take a look at my solution.

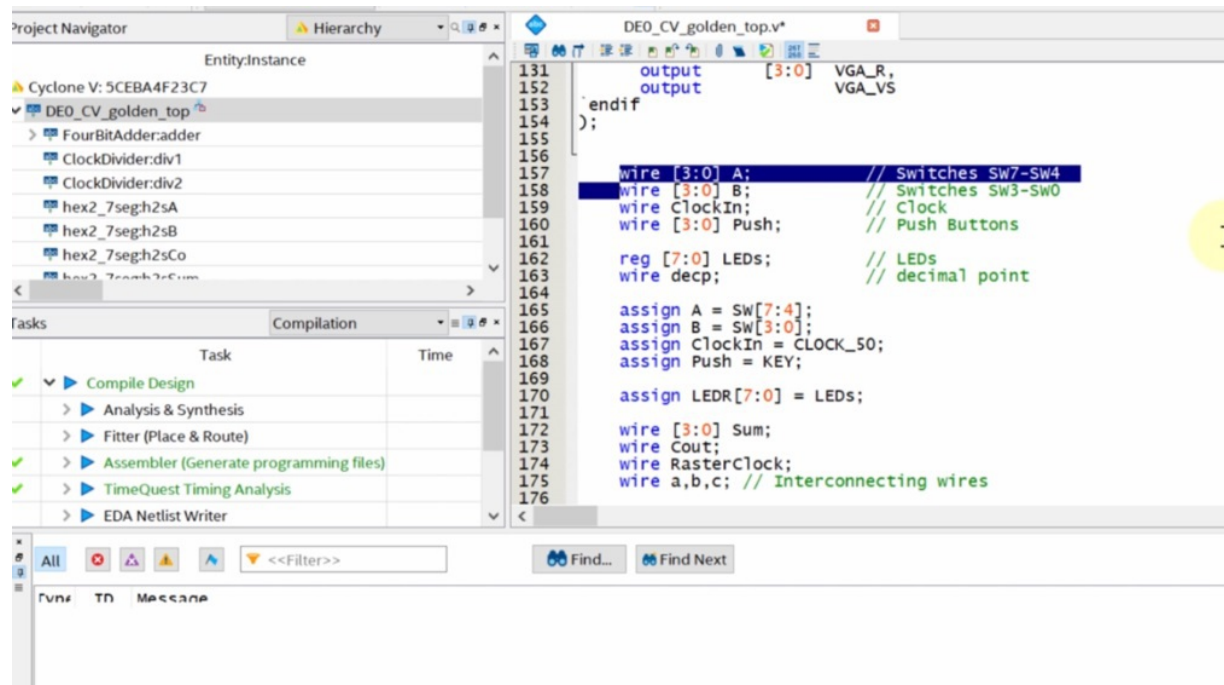
LOOKING AT THE ADDER CODE

Now let me show you my design application. This one it's called either with this plea and here is the cue file which is the project file for quirks that must be what this stands for Quartus project file. And here is my design. Once more let's go to the project navigator at the left. And here is my treat us elements that I am using. And so let me show you the top module as you can see here.

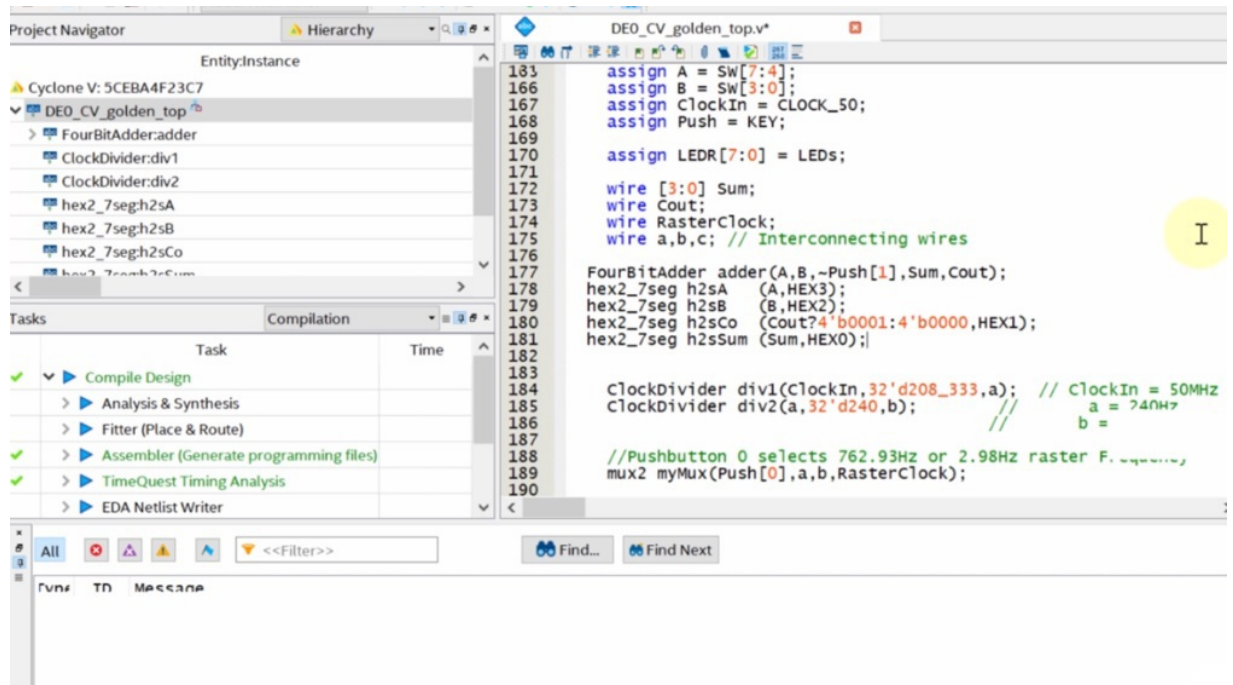


I am not using two of the hex displays but I am using the rest I am using four of them the four digits at the right and I am using the rest of the hardware. I shouldn't use all of it but I am so scrolling down. Now here is my design. You'll see some wires and assignments right here that come from the fact that this code was forwarded from some

other board. I originally made this code for a basis to board and so I decided to include the input output ports as wires here and assigned them with the hard wiring method of a sign in very long so that I can use the definitions for the symbols that are defined in the list.



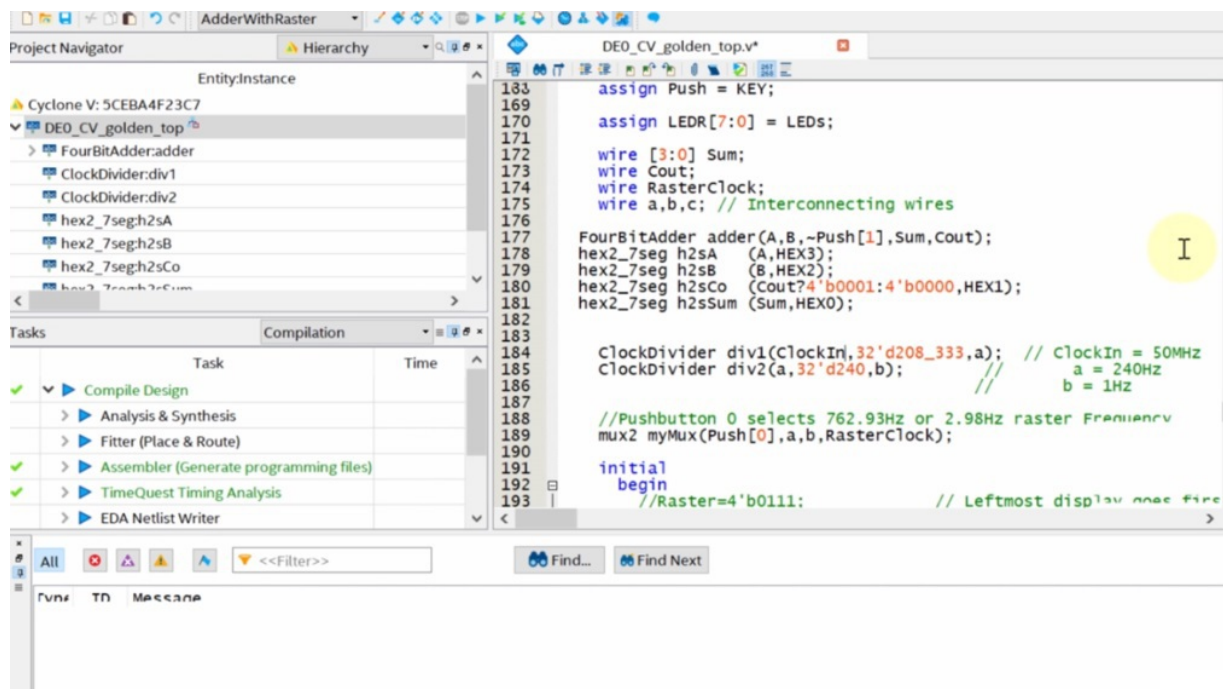
In this design to comply with my name's right here. So here we have it. I have a four wire array named a for variable A then another called B. I have a clock input and this is just to show the frequency of the leading. Then we have the push buttons are ready for push buttons. The LtDs. This one is called LtDs and it's a register. And finally I have something for the decimal point I used in my original design. I'm not using it here. So here are my assignments. I have assigned statements for a which corresponds to switches 7 through 4 in the board. Then I have B which corresponds to switches 3 to zero in the board my wired that was called Glocken is actually the clock with the line in this design and the push buttons. I just defined push are called G. In this design the LCD array I called it LtDs and it's actually Leidy are. And so we have some wires here.



First we have some value which are forbit then the carry out line and these lines Ruster clock and ABC just to show you the rate at which the lower order really is going to be blinking at. So here's my design. Actually the real design is only in these five lines and these lines are the forbit aether that's called rather that uses a B The negated state of pushbutton number one and the outputs are some and C out which is a four bit array. Next I use the hex to 7 segment decoder. I will show you in a bit but it's just like a BCT to 7 segment display driver and it takes in a the hex display number three then the other one takes B to the hex display number two. The next one uses this yellow line. If it is 1 then it sends one in four bits. Or if it's 0 then it's Since 0 and 4 bits to the hex display number one. And finally we have that the sum goes to the hex display number zero. That's it. That's my design.

LOOKING AT THE BLINKY CODE

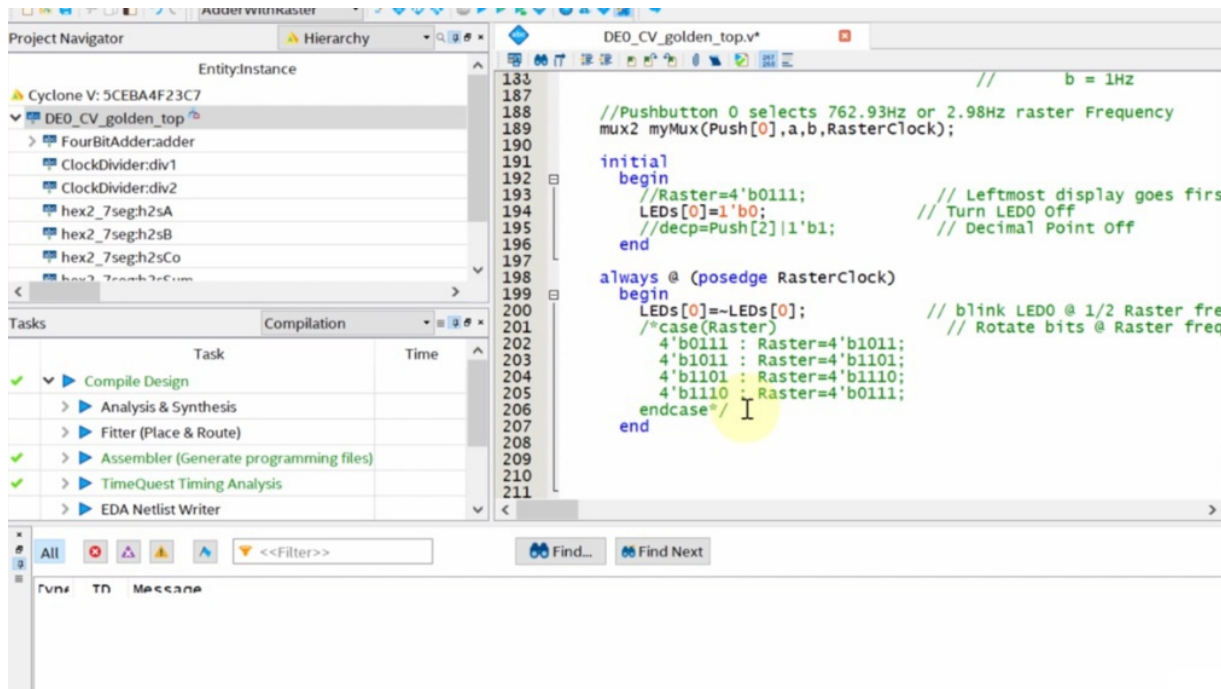
Now for the blinking of the Elodie I have a clock of either here that I'm using to divide the five megahertz line by two hundred and eight thousand three hundred thirty three. So here's a nice thing about very low and it's that you can separate numbers with the underscore character without any meaning.



```
183 assign Push = KEY;
169
170 assign LEDR[7:0] = LEDs;
171
172 wire [3:0] Sum;
173 wire Cout;
174 wire RasterClock;
175 wire a,b,c; // Interconnecting wires
176
177 FourBitAdder adder(A,B,~Push[1],Sum,Cout);
178 hex2_7seg h2sA (A,HEX3);
179 hex2_7seg h2sB (B,HEX2);
180 hex2_7seg h2sCo (Cout?4'b0001:4'b0000,HEX1);
181 hex2_7seg h2sSum (Sum,HEX0);
182
183
184 ClockDivider div1(ClockIn,32'd208_333,a); // ClockIn = 50MHz
185 ClockDivider div2(a,32'd240,b); // a = 240Hz
186 // b = 1Hz
187
188 //Pushbutton 0 selects 762.93Hz or 2.98Hz raster Frequency
189 mux2 myMux(Push[0],a,b,RasterClock);
190
191 initial
192 begin
193 //Raster=4'b0111; // Leftmost display ones fire
```

But this underscore is doing what a comma would when we write numbers by hand and the output goes to the line. It's called a. And later I divide further this line a by 240 into B. What this accomplishes is having two hundred and forty hertz in a and one Hertz in B. Next I use the push button number zero in a multiplexer to either send a or b to this line that is called raster clock and all this mention about

arrester is because in the basic stew board I was doing a arrester. I was turning this place on and off to show just one digit at a time in one display at a time and do a sweep of all of those values to make the olution with persistence of vision that all of the displays were on at the same time.

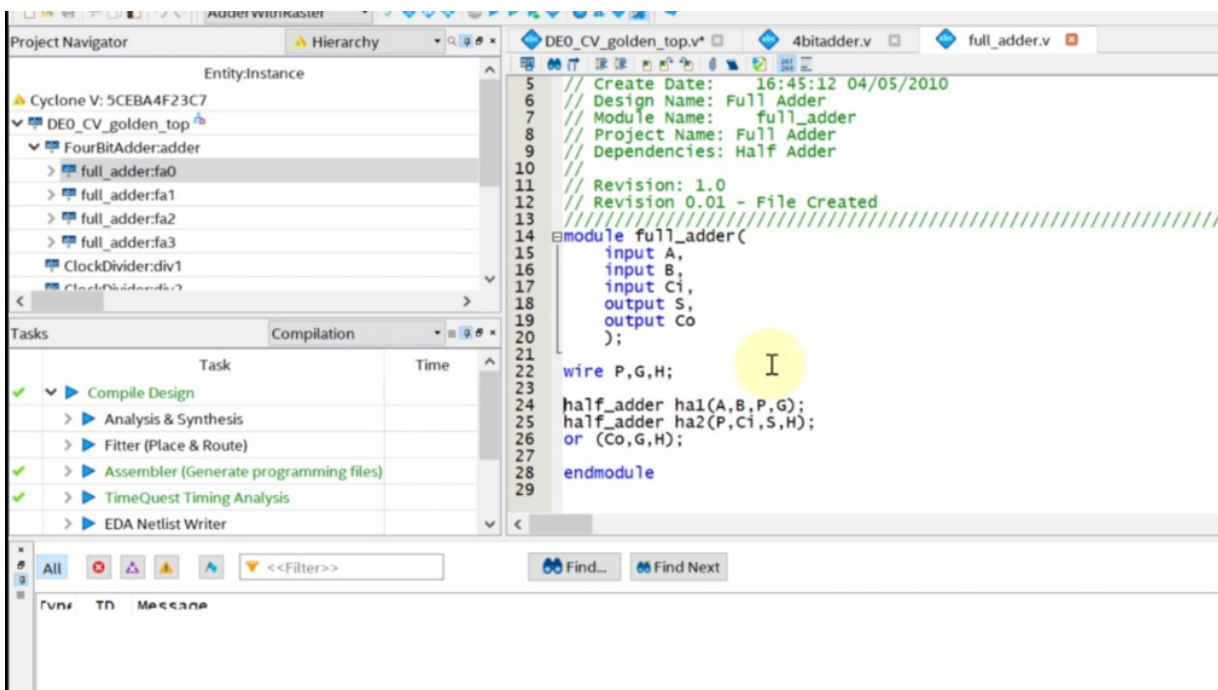


```
183 // b = 1Hz
184
185 //Pushbutton 0 selects 762.93Hz or 2.98Hz raster Frequency
186 mux2 myMux(Push[0],a,b,RasterClock);
187
188
189
190
191 begin
192 //Raster=4'b0111; // Leftmost display goes first
193 LEDs[0]=1'b0; // Turn LED0 Off
194 //decp=Push[2]|1'b1; // Decimal Point Off
195 end
196
197
198 always @ (posedge RasterClock)
199 begin
200 LEDs[0]=~LEDs[0]; // blink LED0 @ 1/2 Raster freq
201 //case(Raster) // Rotate bits @ Raster freq
202 4'b0111 : Raster=4'b1011;
203 4'b1011 : Raster=4'b1101;
204 4'b1101 : Raster=4'b1110;
205 4'b1110 : Raster=4'b0111;
206 endcase //
207 end
208
209
210
211
```

So I was doing that at 240 hertz or at one Hertz controlled by push button number zero. The only thing I need to do for the blinking of the D is to have this initial block actually setting some value on the the number 0. And finally in an always block sensitive to the positive h of the rest or Gluck's signal I can simply just toggle the state of this Elodie. Notice that these lines right here this late state machine lines in case what they are doing is exactly that restor I needed. I am moving a 0 around four lines which were the anodes for the LCD displays. It doesn't matter here. So let me go ahead and just remove these lines.

LOOKING AT THE INSTANTIATED MODULES CODE

Now let me show you some of those instantiated devices right here. First we have the forbit eather and it's very similar to the one we designed in the project about very long.

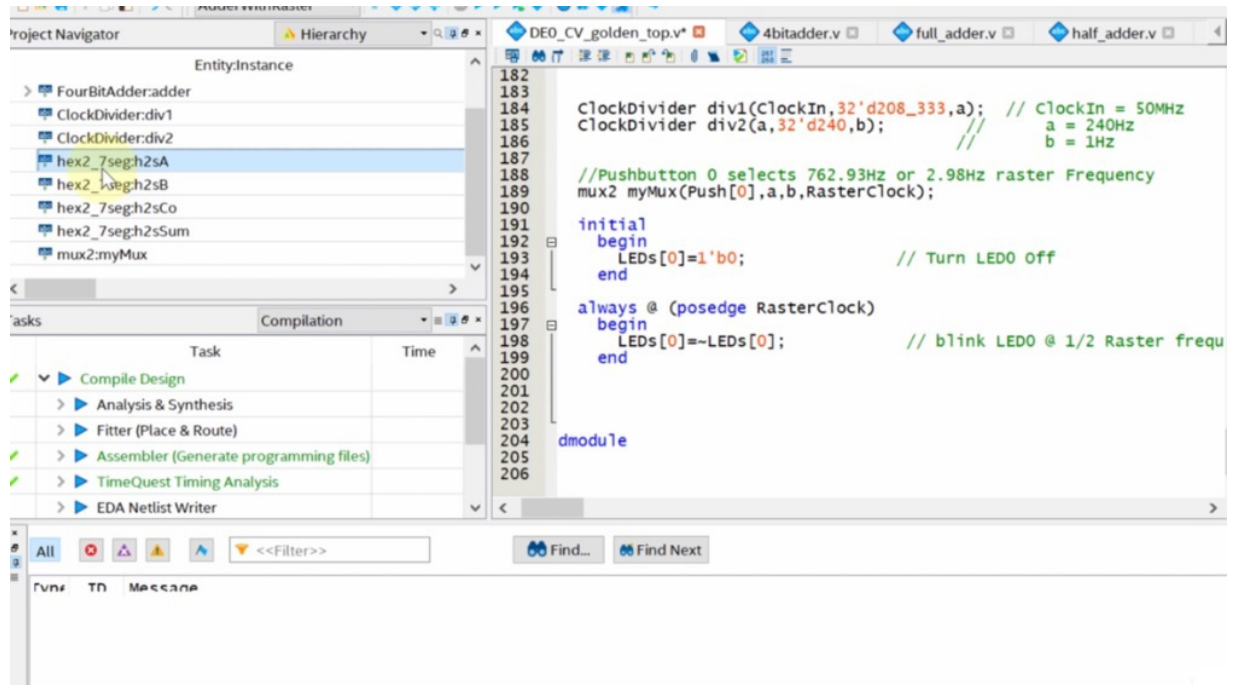


But this one uses only four others. So this one has the Kerry input and it instantiates four others guess what those are well these are the same for others we designed earlier they use half ladders and or gate. Let's see what the half hour is. As you may remember this is just next door and an AND gate No the clock divider remember this model was used to divide the frequency.

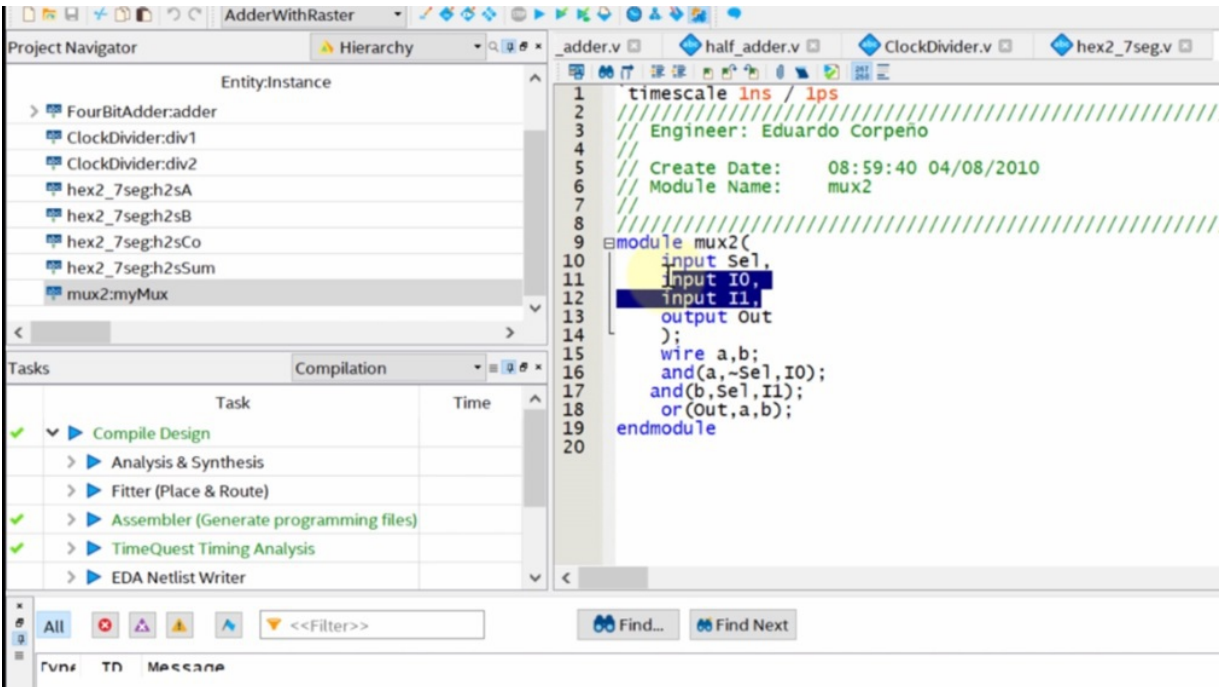
```
1 timescale 1ns / 1ps
2
3 // Module Name: ClockDivider
4
5 module ClockDivider(
6     input ClockInput,
7     input [31:0] ratio,
8     output reg ClockOut
9 );
10 reg [31:0] counter;
11 initial
12     begin
13         counter = 0;
14         ClockOut = 0;
15     end
16 always @ (posedge ClockInput)
17     begin
18         counter = counter + 1;
19         if (counter >= (ratio/2))
20             begin
21                 counter = 0;
22                 ClockOut = ~ClockOut;
23             end
24     end
25 endmodule
26
```

The screenshot shows an IDE window with a project hierarchy on the left and a Verilog code editor on the right. The project hierarchy includes a hierarchy of modules like FourBitAdder, ClockDivider, and hex2_7seg. The code editor shows the implementation of the ClockDivider module, which takes a clock input and a ratio, and outputs a clock signal. The code includes a 32-bit counter that increments on the positive edge of the clock input and toggles the output when it reaches half the ratio.

Well it takes an input line a ratio and provides a single line output which is the clock out line. So what is it doing. It has a counter a 32 bit counter initially it's at zero and the output line is at zero and whenever it receives a positive edge of the clock input then it increases the counter. And when the counter has reached one half the ratio it will toggle the state of the output and reset the counter. What it does is exactly dividing the frequency of the clock input by the ratio we set here.



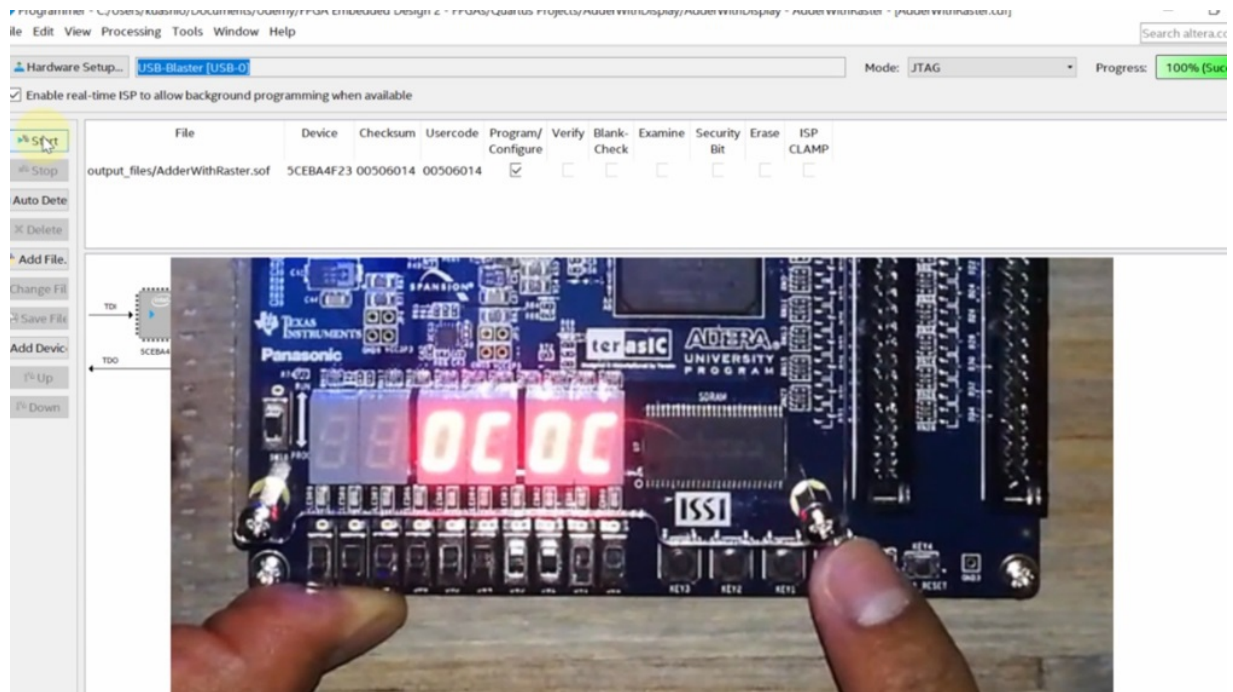
That's why in our code we divided the clock in by two hundred and eight thousand three hundred and thirty three that yields 240 Hertz if the input is at 50 megahertz. The same goes for the next line here where we divided 240 Hertz by 240. That means one Hertz moving on with the instantiated modules. Here we have the hex to 7 segment display decoder. And what it does is that it takes in a 4 bit number and outputs a 7 bit number. I wrote this one by simply assigning values based on these four wires DCB and a which are the bits in the in array the input array.



No the output array. I have a signed statements here also for separate variables g f e DCB A which correspond to the segments of the display. And as you can see here I am performing assignments assignments implemented as logical functions or boolean algebra functions if you care to take a look at this code and verify that it's OK. Please do. And finally we have a multiplexer. This is a two to one multiplexer. Nothing fancy here. We have a selection input just one bit. We have two inputs and the output and this one is implemented with exactly three gates and gates and or gate. So let's go back to our top model and just compile by the way in the bottom right corner you can see the progress. Right now it's at 51 percent. And how long it's taking it's been one minute and 13 seconds. Now and it's done. It took one minute and twenty seven seconds.

PROGRAMMING THE ADDER INTO THE BOARD

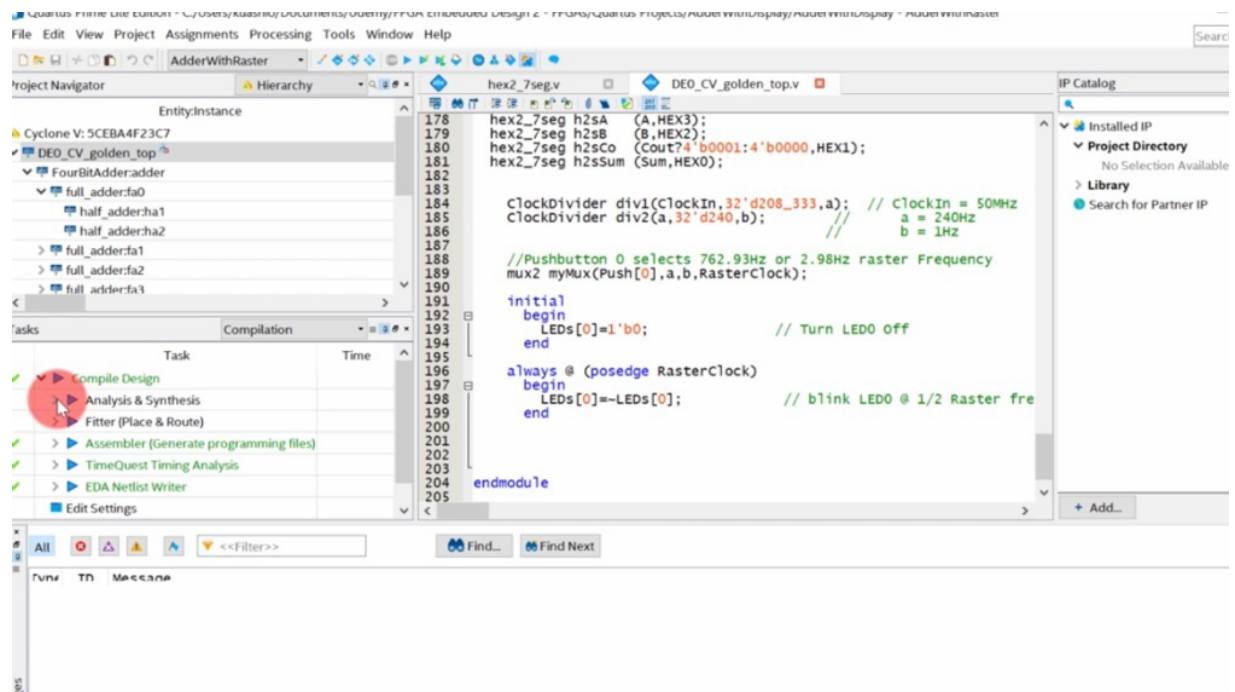
Let's go to the programmer and let's just program the FPGA this is way faster. So let me do that and let's look at the board. There it is first let's look at the LCD as you can see it is blinking at something that looks like one hertz. And if I press the rightmost push button the D will seem a bit dim That's because it's blinking at two hundred and forty Hertz. If I released the button it will go back to one Hertz and that's just a perk for this application. We are really interested in the other that's implemented here.



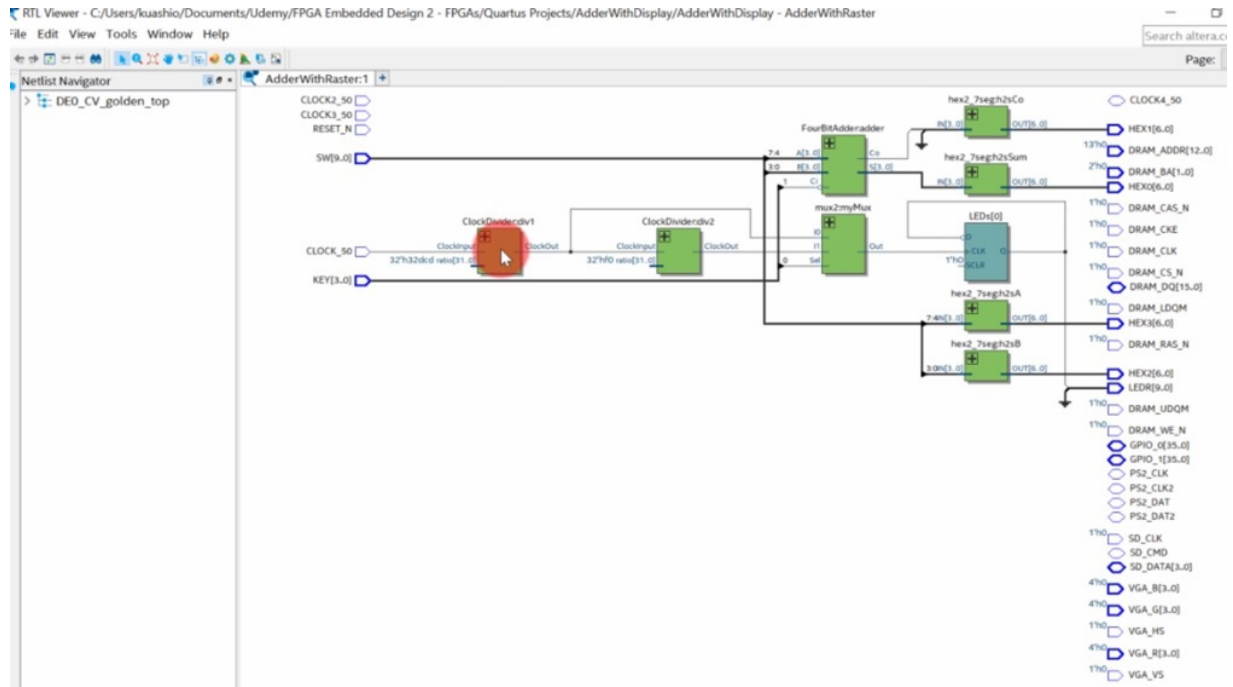
So what that says is that 0 plus C equals zero C.. And that's because I have these two switches on. So let's turn them off. So here's the some as you can see we have zero plus zero equals zero. Let me turn on the least significant bit of being that will be zero. Plus one equals zero. One let's turn that value of A into a. That means I have to turn on this line eight plus one equals zero nine no let's say eight plus 10. Remember this is hexadecimal. So eight plus 10 looks like this 8 plus a which is 12 in hexadecimal 8 plus 10 is 18 in decimal and in hexadecimal that is 16 plus 2 that's 18. And if we turn all of the switches on we will get F plus F equals 1 E which is you know correct. So once more this is just a demo application so that you can get a feeling of what you can accomplish with this board and sky's the limit. You can do just anything you want as long as it's a digital circuit. And as long as it fits inside this FBD. But trust me your designs will fit into this cycle in five FPGA.

SCHEMATIC RTL DEMO

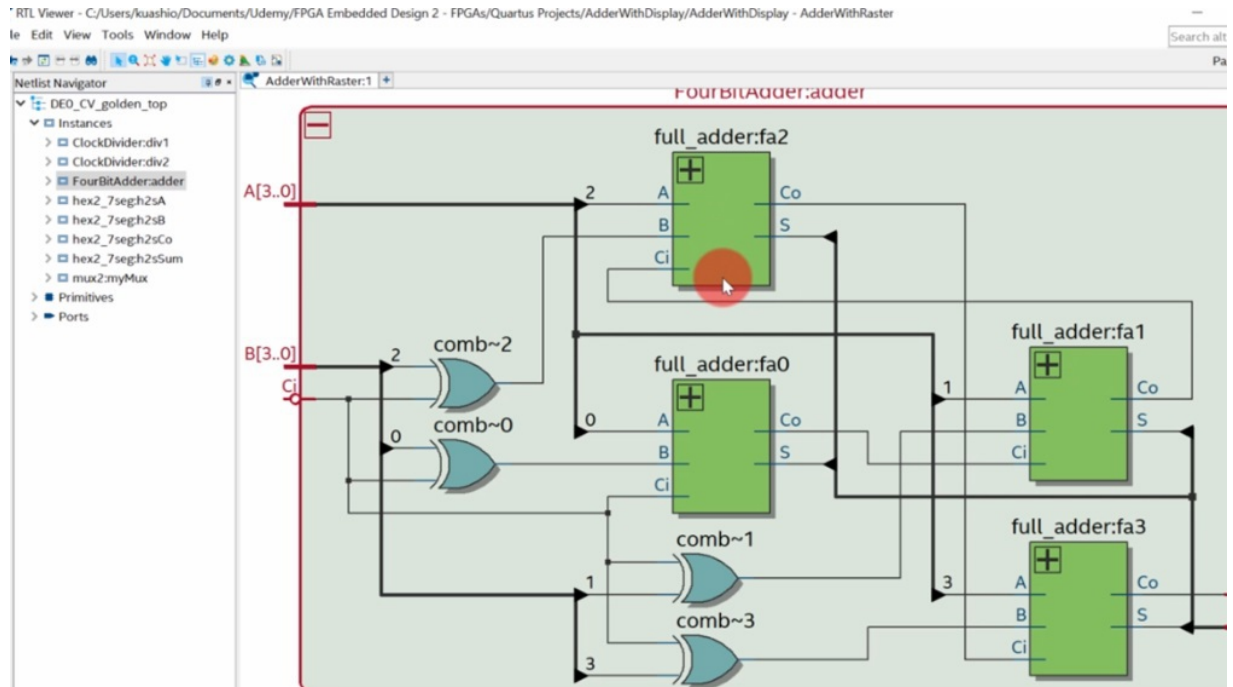
One part of quartus say haven't shown to you is the schematic representation of your designs. So yes here we have our Either our Top Model and this is all very long. But hey did you know that you can take a look at this design in a schematic way.



Well let's do that. So if you go to analysis and synthesis and you open this section you'll see that we have an IOWA assignment analysis a design assistant. So if you look at the analysis and synthesis part here we have the netlist viewers and let's take a look at the RTL view we're well as you can see here.



This is a schematic diagram of our system. Here we have the clock divider. If you double click you'll see what was implemented inside. Here's the counter. It has an other to a constant of one and Kerry input of zero. And it has a clock output that goes on or off whenever this comparison has true or false output. I'm not sure this is a less than comparator and it compares to the input we sent into it. So this makes complete sense. Let's go back to that Top Model Let me zoom in let's see some other block the main block the bit aether. So let me don't click on the name and here it is. It has a full other. Another for the other another and another. So these are the X or gates that are implemented inside and I'm not sure they enter these extra gates but they are here and look at the successive Kerry inputs and outputs. So we have a carry input here that eventually goes into this fall either this Korean put.



Now this carry output goes into this one and this current output goes into this one and this carry output goes into this one. So this is not the nicest layout or the layout we proposed but it's the exact circuitry we entered. And if we look inside a full adder we have to have others and one or gate that's inside the half either we have the XOR and the AND gate. Let's go back to the main design remember that we have an LCD blinking application. Well here it is. Here's the multiplexer that selects which of the clock dividers outputs will go to its output and the selection line is input the number 0. One thing we didn't see in the demo in the live demo was that the input key number one goes to the carry input of the other adders. We didn't see that working but it does work if you press key. Number one it will add one to the whole sum. So let's go see inside these hex 2 7 segment decoders because I didn't write the code for them I just wrote the boolean expressions and see what's inside. And man look at that. It's just a combinational circuit a very very dense combinational circuit. So there's a lot of power inside an FPGA as you can see. All right. Well that's it for this quick demo of the schematic view.